PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 15/80, 15/66	A1	(11) International Publication Number: WO 93/19431 (43) International Publication Date: 30 September 1993 (30.09.93)

(21) International Application Number: PCT/GB93/00580

(22) International Filing Date: 22 March 1993 (22.03.93)

(30) Priority data:
9206126.6 20 March 1992 (20.03.92) GB

(71) Applicant (*for all designated States except US*): MAXYS CIRCUIT TECHNOLOGY LTD. [GB/GB]; 41 Carlyle Avenue, Hillington, Industrial Estate, Glasgow G51 4XX (GB).

(72) Inventors; and
(75) Inventors/Applicants (*for US only*) : MACKIE, Stuart [GB/GB]; 67 Belhaven Terrace, Wishaw, Strathclyde ML2 7AY (GB). MACKAY, Stuart [GB/GB]; 83 Hyndland Road, Glasgow G12 9JE (GB).

(74) Agents: McCALLUM, William, Potter et al.; Cruikshank & Fairweather, 19 Royal Exchange Square, Glasgow G1 3AE (GB).

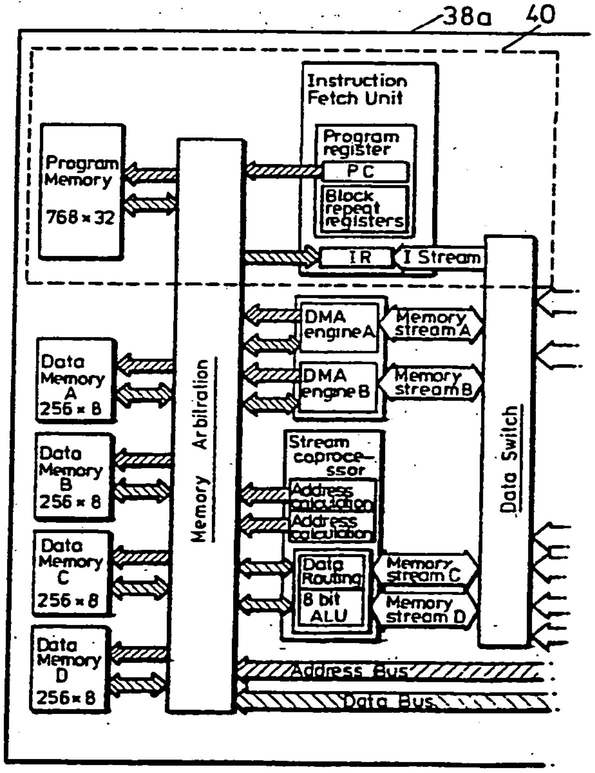
(81) Designated States: AT, AU, BB, BG, BR, CA, CH, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, LK, LU, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SK, UA, US, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: PARALLEL VECTOR PROCESSOR ARCHITECTURE

(57) Abstract

A parallel vector processor (PVP) is described which can be implemented on a single silicon die or multiple dies using single-instruction multiple data (SIMD) or multiple-instruction multiple data (MIMD) architecture containing at leat one data processor with at least one data stream coupled thereto for inputting data and removing results. The PVP apparatus can be incorporated in any suitable communication network topology. In one arrangement this is achieved most readily by providing the parallel vector processor on a single a chip using single-instruction, multiple data (SIMD architecture) containing a plurality of data processors organised into a pipeline with an input/output stream for supplying the processors with data and to remove the results. Although a single input/output stream can be used it is desirable to use at least two streams to facilitate optimum performance. The PVP has applications in neural networks, pattern recognition and a variety of signal processing applications.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	MR	Mauritania
AU	Australia	GA	Gabon	MW	Malawi
BB	Barbados	GB	United Kingdom	NL	Netherlands
BE	Belgium	GN	Guinea	NO	Norway
BF	Burkina Faso	GR	Greece	NZ	New Zealand
BG	Bulgaria	HU	Hungary	PL	Poland
BJ	Benin	IE	Ireland	PT	Portugal
BR	Brazil	IT	Italy	RO	Romania
CA	Canada	JP	Japan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SK	Slovak Republic
CI	Côte d'Ivoire	KZ	Kazakhstan	SN	Senegal
CM	Cameroon	LI	Liechtenstein	SU	Soviet Union
CS	Czechoslovakia	LK	Sri Lanka	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	MC	Monaco	UA	Ukraine
DK	Denmark	MG	Madagascar	US	United States of America
ES	Spain	ML	Mali	VN	Viet Nam
FI	Finland	MN	Mongolia		

PARALLEL VECTOR PROCESSOR ARCHITECTURE

The present invention relates to parallel vector processing apparatus particularly, but not exclusively, for use in neural networks, pattern recognition and a variety of signal processing applications.

In existing prior art signal processing applications, for example, in neural networks, an algorithm is supported whose basic function is to classify a set of input signals as one of a set of possible outputs. The internal para:eters, known as weights, in the algorithm have to be adjusted in a training phase in order to optimise discrimination of the algorithm between possible outputs. This involves initialising the weights to random values, measuring the error in the outputs for a known set of inputs and using a "learning rule" to adjust the weights. After a few hundred iterations through the training set, the weights settle to fixed values, and the training is complete. The network may well still not be able to classify the training set 100% correctly, but if the algorithm was well set up, it will be able to correctly classify a high percentage of inputs that were not in the training set. This procedure is called "generalisation". In a real application, the algorithm can then be used to classify inputs and the resulting outputs can be used as desired. Currently, considerable research effort is directed to finding learning rules that converge on sets of weights using as few iterations as possible whilst, at the same time, being computationally inexpensive.

Most neural network and similar signal processing systems, such as image feature extraction systems, to date have been used in research and have involved programs running on workstations, such as Sun (trademark) or PC. This has been convenient for a number of reasons: many researchers own such a machine before starting work with these algorithms; the algorithms themselves can be programmed readily in a conventional language, such as C or Pascal, and such machines are cheap and readily available. Neural network algorithms have demonstrated the ability to solve some small but difficult problems, especially where there is no known analytical solution and, secondly, it has shown up the limitations of existing workstations to perform the vast number of computations required. Experiments have shown that neural network methods perform better than conventional statistical methods and, more importantly, the solution is obtained without detailed analysis of the problem, in a fraction of the time than would otherwise be possible.

It is now well established that computing power in existing work stations is insufficient to allow neural network and similar signal processing algorithms to be used in many real-time applications, particularly in the fields of image analysis, speech analysis and process control. Typically the amount of information to be processed is of several orders of magnitude greater than can be dealt with by an existing workstation. For example, in the field of image analysis, it is well known that many techniques have been developed for image analysis, but one of the main remaining difficulties is that although the calculations are essentially quite simple, each image contains a large amount of data and the calculations have to be performed at every point in the image. For a typical image containing 1000 x 1000 pixels, each of the million pixels in the image is an input to the algorithm. Networks in image-processing are highly structured and have highly restricted connectivity. Units in each layer are conceptionally arranged in a two-dimensional grid corresponding to the shape of the input image. The learning process is simplified by restricting the configuration of the connections between layers. In image-processing, users typically assign a specific task to each layer of units; the units in the first layer performing averaging for noise reduction, units in the second layer identifying features such as edges, oriented lines and line ends at each point and, in the third layer, identifying composite objects such as combinations of the second layer features.

Therefore, as diagrammatically shown in Fig. 1 an image-processing network and algorithm could contain an input layer having a million input units (1K x 1K pixel images), a first hidden layer with a million units each with weights corresponding to 7 x 7 array of units centred on the corresponding point on the input layer, a second hidden layer containing 50 intersecting planes of units of a million units connecting to 11 x 11 array of first hidden layer units with each plane identifying positions of a particular feature in the input image, and an output layer containing 10 parallel planes of units of a million units each with 100 connections to second hidden layer, each plane identifying positions of a composite feature in the image.

It will appreciated that the total number of multiply-accumulator operations to be performed in the processing of a single image is about 7 billion (1000 + 6000 + 49 x 10^6 input pixels). The structure of the algorithm is identical for each input pixel because the same features may occur at any point and the problem partitions into repeatedly applying the same small neural network algorithm at each point in the input stage.

Although the number of connections in this algorithm is large, it is an algorithm capable of distinguishing only simple features in the image. A comparison of dealing with such amounts of data by a conventional Sun SPARCstation and an Array Processor containing a digital signal processor such as a TMS 320C30 chip (Texas Instruments), a Parallel Array based on an Inmos transputer chip with 100 transputers and a proposed neural network chip indicates, in theory, that the workstation would process the data in about 4 hours, the array processor would perform the task in about 6 minutes, the parallel array in 47 seconds, whereas image processing by the proposed neural network processor would take about 5 seconds.

Consequently, special hardware has been in development to implement neural network-type algorithms for use in a variety of applications such as optical character recognition for use with post codes, printed text, signature verification, handwritten text, medical applications such as X-ray and ultrasonic image analysis, air and space applications such as radar image analysis and satellite image analysis, manufacturing in the fields of process control and automatic inspection and financial services such as loan risk evaluation and stock and currency trend analysis.

In a proposed neural network design it has been suggested that special purpose chips could provide a better solution for practical implementations of neural networks and it is documented that special purpose chips employing parallelism are required for fast calculation of neural network algorithms. In particular, in a paper entitled "Implementations of Neural Network Models in Silicon" by Mackie, Graff & Schwartz, Neural Computers (1988) Ed. R. Eckmiller and C.V.D. Malsburg pub. Springer, Verlag, Heidelberg, a chip is described which is based on a digital pipelined processor in which the pipeline architecture is described as including identical vector processing units which perform a single fixed instruction and the input and stored vectors have binary value elements. The chip calculates the largest five vector products using the processors, each attached to one stored word. A method of sorting the results into rank order as they are put into the result pipeline is described and the ability to cascade chips to increase processing power is mentioned. In this prior art arrangement the chip architecture was designed to be able to implement a single algorithm that is used in pattern recognition, image-processing and neural networks. The common feature of these algorithms is that for each input, an operation is performed repeatedly using several sets of coefficients known as weights, (also known as kernels, feature vectors or templates). This architecture exploits the fact that only the input data changes rapidly while the weights are fixed and stored in the chip. The design parallelises the calculation by providing a separate processing node for each stored weight-vector and bit-serial arithmetic is used wherein the data is processed one bit at a time rather than a conventional method of processing all the bits in the data word in parallel. With all data paths on the chip being 1-bit wide this design leads to compact processors, many of which can be fitted in a single chip, and the basic prior arrangement is depicted in Fig. 2.

In the prior art proposal, 50 identical processors lie along a data path along which flows input data, a list of results and a number of control signals. The data path is pipelined so that data is presented to each processor in turn. The input data typically consists of a few hundred bits representing a vector. There are 50 processors on a chip, each of which contains a memory block in which is stored a weight vector, whose format is the same as the input vectors. As an input vector passes through a processing unit, corresponding elements of the input and stored vectors are operated on. After the last element of an input vector passes out of a processor, the complete result is normally placed on to the result pipeline, after the result from the previous processor. After the last processor, the result pipeline contains a vector whose elements are results from all the processors in the pipeline. Input vectors can be presented contiguously because the results have their own data path and processors are re-set without requiring any dead time. Communication on to and off a chip is done in 16 bit parallel so that clock speeds on the circuit board are kept low.

One algorithm which is dealt with is classification where the basic operation is multiply-accumulate, but the result vector is modified as it passes through each processor to contain only the largest N results so far.

At this end of the pipeline, the list contains the largest N values overall. In order to identify which stored vectors generated the results, each processor carries a programmable "tag" that is passed to a list with the accumulated value during the update. Because only the top N values and tags are passed down the chain, the pipeline can be extended to contain as many stored vectors as necessary and with no loss of throughput, because the result vector need never become longer than an input vector.

Among other algorithms which map onto this architecture are convolution algorithms in which the processor performs a multiply-accumulate function and the results from each processor are placed sequentially in the list string. In such a case, the resolution and number of results must be tailored so that the total vector length is less than each input vector. The chip was designed to operate at 100 MHz and produce a list of the 5 best distances with tag strips every 1.3 u.s. with a latency of 2.5 u.s. and was expected to perform 5 billion operations/second, a rate then about 1000 times faster than a conventional computer. The design enabled chips to be cascaded in series if more than 50 vectors require to be stored and the advantage was that when more processing units were added, the input data rate remains constant, while the only penalty is in increased latency in producing a given result; however, this delay is very small.

With the aforementioned prior art arrangement the circuitry is hard-wired and can only perform multiplication, addition and comparison. Thus, the disclosed prior art proposal for parallel vector processing is very specific and cannot perform other desirable mathematical functions such as subtraction and squaring. In image-processing, for example, the formula $Oj= \sum_{i} W_j I_i$ provides a result (O) which indicates the closeness of a signal vector (I) to a particular direction defined by the vector W. A further requirement of image-processing is to find closeness of position or distance which is more useful than the closeness of direction in some circumstances. This requires to data to be analysed on the basis of $O_j = \sum_{i} (I_i-W_i)^2$. This result cannot be achieved with the disclosed prior art vector processing arrangement.

Furthermore, in many applications there is a requirement to adjust the values of the weights stored at each processor to optimise performance; this cannot be done with the disclosed prior art arrangement. Indeed, the circuitry hard-wired would be extremely complex and would result in an arrangement which would be unrealistically complicated and very expensive.

An object of the present invention is to provide an improved parallel vector processing circuit architecture which obviates or mitigates at least one of the aforementioned disadvantages.

A further object of the present invention is to provide a parallel vector processor which allows hardware to be manufactured for interfacing with existing computers to function as a host computer accelerator and as a stand-alone analysis or control system.

A further object of the invention is to provide a modular parallel vector processing apparatus arrangement where extra processing units can be added to provide increased computing power.

This is achieved in its broadest aspect by providing a parallel vector processor (PVP) which can be implemented on a single silicon die or multiple dies using single-instruction multiple data (SIMD) or multiple-instruction multiple data (MIMD) architecture containing at least one data processor with at least one data stream coupled thereto for inputting data and removing results. The PVP apparatus can be incorporated in any suitable communication network topology.

In one arrangement this is achieved most readily by providing the parallel vector processor on a single chip using single-instruction, multiple data (SIMD architecture) containing a plurality of data processors organised into a pipeline with an input/output stream for supplying the processors with data and to remove the results. Although a single input/output stream can be used it is desirable to use at least two streams to facilitate optimum performance.

According to one aspect of the present invention there is provided parallel vector processing (PVP) apparatus for processing data, said parallel vector processing apparatus comprising,

at least one data processing means adapted to be coupled to at least one stream of data, said stream carrying data into and through said data processing means and carrying results from the data processing means after operations have been performed on the data, said data processing means having an internally addressable data memory, said internally addressable data memory having a plurality of stored coefficients for acting on the data in said stream, said data processing means including calculation means whereby operations are performed on the data in said stream by said coefficients retrieved from said data memory or on the data from said stream itself or on the coefficients themselves to provide a result which is passed to result stream and/or stored in the said data memory, and an instruction processor means coupled to said at least one data processing means for supplying instructions to the data processing means to perform operations on data routed to said calculation means, the result of said operations being routed to said stream for output or stored in data memory.

Preferably, said at least one data processing means are coupled to at least two pipelined streams of data, one of said streams carrying data into and through said at least one data processing means and the other stream carrying results from said at least one data processing means.

Preferably, the apparatus includes a plurality of data processing means each of which is coupled to each of said at least two pipelined data streams, and said instruction processor being coupled to one of data processing means. Conveniently, this is the first data processing means in said pipeline. In alternative arrangements, several or all of said data processing means may include an instruction processor.

Conveniently, the or each data processing means includes data routing means coupled between the data streams, the instruction processor and said internally addressable data memory, the arrangement being such that execution of said instructions is controlled by said data routing means whereby operations are performed on said data in said calculation means by said coefficients retrieved from said internally addressable data memory and the results of said operations are placed on said result stream or stored in data memory.

Conveniently, the data routing means includes a network switch means coupled to said data streams, data switch means, said calculation means and memory arbitration means.

Preferably, also a parallel vector processing chip is provided having a plurality of said data processing means and an instruction processor formed on a single silicon die which uses serial single instruction-multiple data (SIMD) architecture. Conveniently, 32 data processing means are formed on a single PVP chip (designated PVP 32) and a parallel vector processing (PVP) device may have several of such chips, for example, 64 PVP 32 chips.

Conveniently, each data processing unit includes at least one DMA engine and controller for transferring blocks of data to and from the data memory in each data processing units.

Advantageously, the data streams are bidirectional. Alternatively, the PVP could be located in an arbitrary communications network.

Conveniently, in use of the PVP, the use of multiple banks of memories in the calculation unit allows simultaneous read/write of data to and from memory.

Conveniently, the PVP is implemented in a system accelerator for interfacing with a bus-interface of existing workstations for increasing the speed of processing of said workstation, said accelerator having a PVP consisting of a plurality of such PVP chips, a data input port and a data output port, buffer memory means coupled to said input port and said output port, and data preparation means and post-processing means coupled between the PVP means and said buffer memory and an internal interface also coupled to the buffer memory interfacing with the bus-interface of the workstation.

According to another aspect of the present invention, there is provided parallel vector processing (PVP) apparatus for processing data in parallel, said PVP apparatus comprising:

a plurality of data processing means adapted to be coupled to at least two pipelined streams of data, one stream being coupled to each data processing means for carrying data to each of said data processing means, the other stream being coupled to each data processing means for carrying results from said data processing means after operations have been performed on the data, each of said data processing means having an internally addressable data memory having a plurality of coefficients stored therein for acting on said data in said one data stream, said data processing means including calculation means for performing operations on said stream by said coefficients retrieved from said data memory or on the data from the stream itself or on the coefficients themselves to provide a result on said other data stream and/or stored in the said data memory,

instruction processor means coupled to one of said data processing means for supplying instructions to all of said data processing means to perform operations on data routed to the calculation means,

the or each data processing means including data routing means coupled between said at least two data streams, said instruction processor and said internally addressable data memory, the arrangement being such that execution of said instructions is controlled by said data routing means whereby in each data processing means, coefficients are retrieved from said internally addressable memory and operate on data from said one data stream to provide a result, said result being passed to said other data stream or stored in data memory, said data processing means performing said operations simultaneously whereby input data is processed in parallel in said PVP apparatus. The PVP is conveniently implemented on a single silicon die.

According to another aspect of the present invention there is provided a method of processing data, said method comprising the steps of:

storing a plurality of coefficients in an internally addressable data memory of at least one data processing means to operate on a stream of input data,

receiving said stream of input data from said at least one data pipeline and performing operations on said data by said coefficients from said data memory in said data processing means in accordance with instructions from instruction processing means coupled to said at least one data processing means to provide a result, routing said result to said stream for output or storing the result in data memory; and

repeating the procedure for a plurality of data elements in said data stream.

Conveniently, data is input on one data stream and the results are output on a second data stream.

Preferably, a plurality of data processing means are cascaded together and are coupled to said at least two streams to form a pipelined processor, said instruction processor being coupled to one of said data processing means, whereby the method outlined above is repeated in each data processing means for each element in the input data stream, so that simultaneous operations are performed on a plurality of different input data elements using coefficients from respective data processor memories, the results of said simultaneous operations being output onto said second pipelined data stream or stored in data memory.

Conveniently, the method is used to provide vector scalar product calculations for use in image processing, weight updating calculations for use in neural network applications or wavelet transformations for image compression,

According to another aspect of the present invention, there is provided a method of processing data in parallel to achieve faster processing of input data, said method combining the steps of,

providing a plurality of data processing means and coupling at least two pipelined streams of data to each of said data processing means, each data processor having an internally addressable data memory, storing a plurality of vector coefficients in each data memory,

passing data along one of said pipelined streams of data so that all data in said stream passes through each data processing means,

retrieving stored coefficients from said respective data memory under the control of instructions from an instruction processor coupled to one of said data processing means and operating on said data in said data processing means with said coefficients to provide a result,

outputting the result of each operation of said coefficients on said data onto said other of said data pipelines or storing the result in data memory,

said operations in a plurality of said data processing means being carried out at the same time whereby the serial inputted pipelined data is processed in parallel.

These and other aspects of the invention will now be described with reference to the accompany drawings in which:-

Fig. 1 is a diagrammatic representation of the input and output layers in an image processing network depicting how a very large number of calculations require to be performed for a single image.

Fig. 2 depicts the architecture of a prior art digital pipelined processor having a plurality of identical processing units for performing a classification algorithm where each new result is compared with those already in the pipeline and the larger result inserted into the pipeline;

Fig. 3 depicts to perspective view of a Sun SPARCstation including a network accelerator in accordance with an embodiment of the present invention;

Fig. 4 is a schematic block diagram of a network accelerator sub-system for use with the accelerator as shown in Fig. 3 in accordance with the present invention;

Fig. 5 is a schematic block diagram of part of the network calculation unit shown in Fig. 3 in accordance with the present invention and depicting one data processing unit and an instruction processor;

Figs. 6a and b depict schematically a set of registers in the data processor shown in Fig. 5 and the organisation of one of the 24-bit accumulators A and B;

Fig. 7 depicts the organisation of the data RAMs in the local memory of each data processor;

Fig. 8 depicts the configurations of control blocks in a DMA controller and how these are used to reference a block of data in memory;

Fig. 9 depicts schematically the set of registers in a DMA controller which is required to support the DMA control blocks;

Fig. 10 is the schematic diagram of a configuration register block which also depicts the mapping DMA controller registers in memory shown in Fig. 9;

Fig. 11 depicts the basic instruction set for the data processor shown in Fig. 6;

Fig. 12 is a schematic diagram of a pipelined execution instruction;

Fig. 13 is a schematic view of the integer unit shown in Fig. 5 depicting data flow paths through the integer unit during calculation of the scalar product of two vectors;

Fig. 14 is a flow chart depicting the sequence of events in the calculation of the vector scalar product in the integer unit shown in Fig. 13;

Fig. 15 is a table of internal processor values appearing in the outputs of the components of the processor shown in Fig. 5 during scalar product calculation;

Fig. 16 is a schematic view showing how multiple memory banks are used to handle the data flow through the integer unit of Fig. 5 during calculation of weight updates.

Fig. 17 is a flow chart depicting the sequence of events in the calculation of the weight updates in the integer unit for the data flows in Fig. 9;

Fig. 18 is a table of internal processor values appearing at the output of the components of the processor shown in Fig. 5 during weight update calculation;

Fig. 19 depicts a section of a pipeline three processors cascaded together;

Figs. 20a and 20b depict diagrammatically the connections required between adjacent processors in order to cascade processors;

Fig. 21 depicts the pin assignments for each pin in the two data processor shown in Fig. 19 to allow the processors to be cascaded, and

Fig. 22 is a schematic diagram of a circuit board containing 8 PVP chips in 2 groups, S-bus interface circuitry, PVP control and buffer memory.

Reference is first made to Fig. 3 of the drawings which is a diagrammatic view of a Sun sparcstation (trademark) system generally indicated by reference numeral 10, consisting of a Sun monitor 12 and a Sun CPU 14 on which is stacked an accelerator hardware unit 16 in a look-alike box and which interfaces with the Sun apparatus via connection with the Sun's internal-bus via connectors at the back as will be later described in detail. As will also be later described, the network accelerator is transparent to the user and the user is only aware of much faster processing power available.

Reference is now made to Fig. 4 of the drawings which is a schematic block diagram of a network accelerator sub-system incorporated into the unit 16 shown in Fig. 3 and which is shown interfaced to the Sun S-bus interface 22 which, in turn, is shown coupled to the host Sun microsystem S-bus 23. It will be appreciated that the network accelerator sub-system 20 is external to the host and receives input data from the host computer, or directly from external sensors, via input port 24 into buffer memory 26. The data is pre-processed in a data preparation unit 28 before being passed to the network calculation unit generally indicated by reference numeral 30, as will be later described in detail. Once the data has been processed by the network calculation unit, it is returned to a post-processing unit 32 to remove redundant information before being stored in the buffer memory 26. The post-processed results in the buffer memory are then fed to the memory of the host computer via the internal interface 34 and S-bus interface 22 for analysis or can be fed via the output port 36 for further processing by other hardware (not shown). An alternative data path is to avoid the buffer memory 26. In this case, data passes from the input port 24 to the pre-processing unit 28 and data from the post-processing unit 32 is passed directly to the output port 36.

In the data preparation unit 28, four slots are provided for pre-processing modules. The pre-processing units perform operations such as simple arithmetic pre-processing of data, for example normalising or thresholding, noise reduction and "moving window" generation. This is useful where the neighbourhood of every pixel and the image stored in buffer memory is sent as an input set. For example, in image processing, images from a camera are usually 1024 x 1024 pixels in size and, consequently, the whole image cannot be analysed simultaneously. Consequently, it is normal to process small "patches" surrounding each pixel in turn. Patch sizes depend on the application so the unit is programmable to generate patch sizes of MXN where $1 \leq M,N \leq 256$. An image would be loaded into the buffer memory either from the host or through an external input port, patches are generated for each input pixel in turn and fed into the network processing unit. The post-processing unit 32 performs post-processing requirements, such as thresholding, normalisation and re-ordering, and for each set of inputs, the output from the neural processing unit has a list of the highest few output values obtained in the output layer, together with tags identifying which output unit held the value. In some cases it may be desirable to strip the output leaving only the top tag and/or its value, pass only the top few values and tags, or pass only those values greater than some threshold. It also has the facility to store coordinates of input pixels along with its outputs and this is useful if a threshold is used and high value outputs are sparse.

The buffer memory 26 contains up to 64 megabytes of memory in two banks and acts as a data reservoir where data is placed by the host before processing and where the results are placed after processing before transfer back to the host for display and analysis.

The host interface is in two sections. The first section consists of a small circuit board 22 inside the Sun sparcstation 10 which connects directly to the Sun s-bus and another small circuit board interface 34 is inside the network sub-system 20. Interface 34 contains input and output buffers, drivers to send data to the external computation unit, and receivers to accept data from the unit. The modular nature of this interface 34 means that other interfaces can be supported without modifying the main processor area.

The network calculation unit 30 is based on a PVP (parallel vector processing) chip formed on a single silicon die which are supplied on small modules similar to single in-line memory modules (SIMM), as used in most work- stations. Each unit contains 8 PVP 32 chips and up to 8 modules may be fitted containing a total of 64 PVP 32 chips.

The network calculation unit 30 consists of a chip which integrates multiple, programmable fixed point digital signal processors in a single silicon die. The architecture is single instruction, multiple data (SIMD) architecture containing 16 data processors 38 organised into a simple pipeline with 4 byte-wide I/O streams A, B, C and D supplying the processors 38, one of which is shown in Fig. 5, with data and to remove results. The I/O streams A, B, C and D are each supported by 2 8-bit uni-directional ports which connect the streams to the outside world. The I/O ports A, B, C and D are first-in, first-out (FIFO) ports and support a number of data transfer protocols to allow the processor to be used in as wide a variety of environments as possible.

Examples of the data transfer protocols are synchronous master and slave, asynchronous master and slave and semi-synchronous master and slave.

As will be described each data processor 38 is based on a fully programmable 8-bit pipelined processor core optimised to process vectors of data and so uses an 8-bit word. Each processor 38 has a local internally addressable memory to store algorithm coefficients and local results from processing data flowing in the I/O streams. The instruction processor 40 is configured to control the overall operation of the network calculation unit 30 executing the programs which control the data processors 38 operating on data flowing in the streams by routing the data through the various calculation elements of the data processor 38. The instruction processor 40 contains normal instruction fetching, using a program counter 45 and also hardware looping 46 by repeating an instruction sequence a given number of times. This instruction processor 40 has 3Kbyte of program memory 48 to store programs and data. The lowest 64 bytes are reserved for control and configuration parameters which control the instruction processors operation. The program RAM is based on a 32-bit word enabling instructions to be fetched in a single cycle.

Reference is now made to Fig. 5 of the drawings which depicts one of the 16 data processors 38 of the unit shown in Fig. 4. It will be appreciated that all such data processors 38 are identical and that the following description is applicable to each of them. The data processor 38 is the first data processor (O) in the set and the instruction processor 40 is connected only to the first data processor (O) as shown in Fig. 4 with a data switch 42 and a memory arbitrator unit 44 of processor 38. The instruction processor 40 is also connected to a JTAG port (not shown) which supports board level testing to the IEEE 1149.1 standard. The JTAG port is used to load the instructions which define the port configuration and protocol.

The data processor 38 consists of a network switch 48 which interfaces with I/O streams A, B, C and D which in turn interfaces with data switch 42 via I/O streams E and F. The data switch 42 in turn is coupled to the instruction processor 40, an integer unit 50 for calculating the scalar product of two vectors as will be described, a DMA Engine 52 and a stream co-processor 54. The DMA engine 52 and stream co-processor 54 are coupled to the memory arbitrator unit 44 which, in turn, is connected to data memories 54A, B, C and D, each of which is 8 x 256 bits. An address bus 56 and data bus 58 are connected between the network switch 48, the integer unit

50, data switch 42 and the memory arbitrator unit 44. The network switch allows any of the I/O streams A, B, C and D to be combined and the DMA Engines 52 (A and B) transferring data to an from data memories 54 and the instruction processor 44. The stream co-processor 54 is used to sort data in memory. The integer unit 50 contains 2 sets of registers; sixteen 8-bit registers (2 stream registers and 14 general purpose registers) and two 24-bit accumulators A and B. The stream registers are special purpose and connect the integer unit 50 to the data switch 42 removing the need to execute explicitly load and store instructions. The remaining general purpose registers are used for general purpose processing and hold either data or memory addresses. The integer unit also contains an 8-bit ALU, an 8-bit Array multiplier and an 8-bit barrel shifter to allow scalar product calculations as will be later described in detail.

The 24-bit accumulators A and B are organised, at the program level, as 16-bit registered pairs A(0), A(1) and B(0) and B(1) respectively with the most significant byte used as a status/control word as best seen in Fig. 6a and c.

It will be appreciated that the accumulator precision can be increased by cascading both 24-bit accumulators together using the carry paths to form a single 48-bit accumulator. The cascade bit in the status word, controls whether the carry of an accumulator is propogated to the other. Normally accumulator B is cascaded to A, with B holding the most significant word. However, for symmetry accumulator A can be cascaded to B. It will be understood that cascading both accumulators so the carry paths form a complete circle yield unpredictable results. Any of the status files can be cleared by writing of one to the appropriate bit position.

Each data processor 38 contains 2 kilobytes of local memory which is physically organised as the 4 x 256-byte data RAMs 54a - d. Each processor 38 has a 2 kilobyte address space, which in addition to the data RAMs, contains a set of configuration registers which control the operation of the processor 38. Each data RAM is organised into 64 x 32-bit words using little-endian addressing as shown in Fig. 7. The RAM supports word, half word and byte accesses, though half word accesses must be aligned on memory word boundaries.

Block transfers of data between the data RAMs and either of the I/O streams data processor or instruction processor 40 is performed using memory streams A, B, C and D which are simply the DMA channels used to perform these block transfers of data. Four memory streams A, B, C and D are supported, one for each data RAM which each has one of the local DMA controllers 52A,B to perform the block transfer. It should be understood that the DMA channels are classified as memory streams for architectural regularity because the data processor 38 operates on two streams of data, A and B, without differentiating whether the data comes from the outside world or from local memory. The DMA controllers operate autonomously once they have been configured using a linked list of control blocks as shown in Fig. 8 which allow multiple transfers to be performed without data processor intervention.

Each control block contains 7 parameters which control the data transfer. These parameters are as follows:-

DataPtr: holds start address of data block to be transferred.

ColumnIncrement: 2's complement value added to DataPtr after each byte is transferred.

RowIncrement: 2's complement value added to each RowPtr after a complete row is transferred.

TransferStatus: status/control register for DMA Channel.

RowCount: number of rows left to transfer.

ColumnCount: number of bytes left to transfer from current row.

NewBlockPtr: address of start of next control block in memory.

In addition, the DMA controller contains a set of 8 registers to support the DMA control blocks and these registers are depicted in Fig. 9 which are:-

CurrentBlockPtr: pointer to the current DMA control block, in memory, being executed.

ColumnIncrement: 2's complement value added to DataPtr after each byte is transferred.

RowIncrement: 2's complement value added to each RowPtr after a complete row is transferred.

TransferStatus: status/control register for DMA Channel.

RowPtr: point to the current row address of data in memory.

ColumnPtr: point to the current memory location used in the transfer.

RowCount: number of rows left to transfer.

ColumnCount: number of bytes left to transfer from current row.

It will be appreciated that using both row and column points for data enables the DMA controllers to support both one dimensional and two dimensional block data transfers:-

A 1-D transfer simply transfers a contiguous block of bites between a port and the RAM array.

```
ColAdr  =  StartAdr;
for (ColumnsLeft = ColumnCount; ColumnsLeft >= 0;
     ColumnsLft--)
(
    RamAddress = ColAdr;
    ColAdr = ColAdr + ColumnIncrement;
)
```

A 2-D transfer, is simply a 1-D transfer repeated for a given number of rows:

```
RowAdr = StartAdr;
for (RowsLeft = RowCount; RowsLeft >= 0; Rowsleft--)
(
     ColAdr = RowAdr;
for (ColumnsLeft = ColumnCount; ColumnsLeft >= 0;
     ColumnsLeft--)
(
     RamAddress = ColAdr;
     ColAdr = ColAdr + ColumnIncrement;
)
RowAdr = RowAdr + RowIncrement;
)
```

It will be understood that two-dimensional transfers are useful for handling non-contiguous data and the registers from each DMA controller shown in Fig. 9 are mapped into the configuration register block, as shown in Fig 10, at locations $00 to $IF, although locations $00 to $07 for DMA channel only are shown.

From Fig. 5 it will be seen that each data processor 38 supports the total of 13 data streams: the 4 input/output streams A-D transferring data between adjacent processors (and also between chips), 4 memory streams A-D which support DMA transfers of data to and from each of the data RAMs 54a-d and 2 data streams which transfer data to and from the integer unit 50 and I/O streams E,F which connect the network switch 48 to the data switch 42 and a host stream (I) which connects the data processor 38 to the instruction processor 40 as shown in Fig. 5. It will be understood that the streams are bi-directional though block transfers are only supported in one direction at a time.

Connecting the processor 38 to I/O memory streams A to D to transfer data between the integer unit and the rest of the pipeline removes the need to explicitly load and store data. Data transfer is synchronised with the instruction execution, so that the processor cannot use invalid data or overwrite data waiting to be transferred from the data registers.

Stream routing is performed by the network switch 48 which has a 6 x 6 crossover switch, seen in Fig. 5, which allows data to be transferred from up to 4 source ports, each to one or more destination ports. Stream routing can also perform a limited set of operations on the I/O streams A to D between source and destination ports. Stream routing is controlled by a set of 6 stream controllers which;

selects the streams to route and up to 4 streams can be routed simultaneously;

selects destination ports to route the streams to and a stream can be routed to one or more destinations and

which defines the operations to be performed in the streams being routed.

In respect of the stream operations it will be appreciated that a logical AND function may be performed, that is, destination A = source A and source B. A logical OR function may be performed, that is destination A = source A and/or source B and an exclusive OR function can also be performed. An order data operation may be performed such as destination A = the maximum value of signal from source A and source B, and no operation can be performed in which destination A is = to source A and destination B is = to source B.

The data processor is able to support 2 data formats: 2's complement and unsigned. The instruction set is shown in Fig. 11 of the drawings and the instruction set is based on 32-bit words with 2 basic formats.

Use of the hardware shown in Fig. 5 will be understood with reference to the following operations:

Starting Up

On startup or reset, the default path for data on Stream A is via Stream E and Stream I to IR (Instruction Register) where it is executed. The Instruction Register is 32 bits wide, so four clock cycles will be required to fill it.

Loading a Program

A bootstrapping procedure is used. First code is loaded that will be used to fetch the main program. This is simply achieved by sending a set of Immediate Store instructions where the following two bytes of data are an address and data. Each byte of data is stored at the corresponding address in the program memory 48. The loading program is then executed and has the function of programming one of the DMA engines 52,b to fetch data from the input stream and load it into the program memory 48.

Loading Processor IDs

One of the registers in the integer unit is used to store an ID for the processor. These are loaded as follows. The default routing for data is into the first processor (0) via Stream A and not to be sent to the next processor (1). Thus, the first ID can be loaded using appropriate routing from Stream A into the ID register. After this is complete, the routing is changed to send data to the next processor by changing the state of the data routing register in the stream co-processor in Memory Stream C.

The following code fragment illustrates how this is done:

```
if (Accumulator A = 0)[
   load ID from Stream A into Register 0
   Accumulator A = Accumulator A + 1
   set routing to send data to next processor
[
else [
   do nothing
]
```

Loading Data Memory

The data memories are loaded in turn using the following program:

```
if (Register 0 = Load ID)[
   set DMA engine to load N bytes starting at some address
   load N bytes to data
]
else [
   do nothing
]
```

Executing a Program

A program is executed by sending an instruction that loads the program counter 45 with a string address and transfers data flow into the instruction register 56 to the program memory 48. The instruction processor 40 fetches instructions from program memory 48 according to the contents of the program counter 45, puts them into the instruction register 56, then executes them. A program may contain loops 56 that are controlled by the instruction processor 40.

Ending a Program

When a program is finished it can transfer control back to the input stream and the next program could be loaded through the bootstrap procedure, or if it is known what the next program will be the last act of a program could be to load the next program via DMA.

An instruction execution is completed using a 4 stage clock pipeline:-

IF - Instruction Fetch: The Program Control logic fetches a 32-bit instruction from the Program RAM.

RD - Operand Read: The source registers are read and the instruction opcode is decoded.

ALU - Compute: The specific datapath operation is executed or a memory access is initiated.

WR - Result Write: The datapath result or data from memory is written to the specified destination register.

Fig. 12 is a schematic diagram of a pipelined instruction execution. It will be understood that while pipelined execution reduces the effective instruction execution time from 4 cycles down to 1, a problem exists with interlocks which occur as a result of an instruction which is not written to the destination register until 2 clock cycles after the instruction starts executing. Therefore, if any of the 2 subsequent instructions try and read the results of the preceding instruction, they get invalid data.

Therefore, if an interlock occurs execution of the instruction must stall until the results are valid. The interlock condition is detected by flagging whether the contents of a register are valid and this procedure is known as score-boarding. Whenever a register is written to its score-board flag is set indicating that the contents are valid. When an instruction starts executing the destination registers where results are reserved by clearing the score-board flags thereby preventing another instruction from reading the contents before they are written. The execution pipeline with a separate phase for reading Operands supports the register score-boarding with minimal overhead.

Reference is now made to Figs. 13, 14 and 15 of the drawings in order to explain how the data processor 38 performs calculation of the scalar product of two vectors. Reference is first made to Fig. 13 which is a schematic view of the integer unit 50. It will be understood that the input from the previous processor is Stream A and the output to the data memory 54 is Stream D. The processor wishes to calculate the product of data in Stream A and kernel data in data memory 54. The unit 50 executes instructions under the control of a 4-phase clock. On each phase 1 byte from Stream A and data memory 54 is routed into registers 0 and 1. As the first cycle information from register 1 is routed from registers 0 and 1. On the first cycle information from register 1 is routed from registers 0 and 1 to the multiplier. On the second phase of the clock cycle information from the register is loaded into the calculation units and on the third phase of the clock cycle multiplication takes place and routing for the output. On the fourth phase the results of the multiplication are fed back to registers 2/3 and new data is fetched from the previous processor.

On clock cycle 5 the results in registers 2 and 3 are passed to accumulator A whilst the next multiplication is occurring. The shaded path 60 is the path for loading accumulator A at the end of calculation. It will be understood that it takes three clock cycles to unload the 24-bit accumulator through the 8-bit channel.

Reference is now made to the flowchart of Fig. 14. Accumulated values A,B are fetched in block phase, multiplied in clock phase 2 and accumulated on block phase 3. Once the accumulated value is obtained a check is performed to see if the accumulation is the last input element. If not, the procedure is repeated in a loop manner. If yes, the result is stored and the accumulator is reset. A decision is carried out, in phase 4, as to whether there is another input vector and, if not, the procedure is repeated; if yes, the calculation is terminated.

The scalar calculation of two vector products if based on the following formula:

$$o = \begin{bmatrix} i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} w_{j1} \\ w_{j2} \\ w_{j3} \end{bmatrix} \qquad (1)$$

To facilitate understanding of the calculation procedure the following values are used for the terms in equation (1).

$$I_1 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, I_{\underset{2}{I}} = \begin{bmatrix} 2 & 1 & 5 \end{bmatrix}, I_{\underset{3}{y}} = \begin{bmatrix} 4 & 3 & 1 \end{bmatrix}, W = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \qquad (2)$$

This operation is best explained with reference to Fig. 15 which depicts the internal processor values during scalar product calculation. In particular, the table shows the values appearing at the output of relevant components of a processor during scalar product of a stored 3-element vector with a set of 3 input vectors. From the equation (1) i_1 (123) is the first vector in Stream A. In clock cycle 1, vector (1) from Stream A and vector (4) are stored in their respective registers 0,1. On clock cycle 2, these values are multiplied and the next two vectors 2,5 are read into registers 0,1 respectively. On cycle 3 the multiplied value (4) is stored in accumulator A and the second vector 2,5 are multiplied and, at the same time, vectors 3,6 are read into registers 0 and 1. On the fourth cycle the second multiplied value is added to the first value in the accumulator and the third vector pairs are multiplied (8). On the fifth cycle the third multiplied value (8) is added to the value in accumulator A and the first vectors from the next elements are read into registers 0,1. On the sixth cycle the value in accumulator A is read out into the Stream D co-processor and the second vectors in the next element are read in etc. and so the procedure is repeated. It should be understood that the grey areas marked "X" are no operations (NOPs) between vector calculations and are shown for clarity, although values at these points are undefined. In reality, vectors can be sent through processors contiguously. Stream A receives data from the previous processor (data is passed onto the next processor via Stream B). If the accumulator result occupies more than 1 byte, then up to three clock cycles would be required to unload it.

Thus, this example demonstrates how multiple operations are prepared in each clock cycle to increase the speed of processing. This procedure can be used to cater for much larger vectors, for example 64 bits long, so that the vector processing operations are performed in parallel.

A further example is the calculation of weight updates in neural network learning as is best described with reference to Figs. 15, 17 and 18. In this case, as can be seen from Fig. 16 two data memory banks (0) and (1) are used. The data path is changed on alternative clock cycles so that a modified weight can be written back to one memory bank, whilst weight data is being fetched from the other memory bank. The modified values are passed through registers 4 and 5 in order to delay them for one clock cycle, so that they arrive at the memory during a write cycle.

The striped arrow 62 indicates the path used to load the delta (8) value into register 1 at the start. The flow chart in Fig. 16 shows the sequence of operations and shows how modifies the weight from data memories 0,1 and how the new value is added to the weight and stored.

Reference is made to Fig. 18 which depicts a table similar to that in Fig. 15 except that it depicts internal processor values during weight update calculation. In this example, the calculation for updating weights in the back-propogate neural network learning algorithm is:

$$w_i(t+1) = w_i(t) + \delta_j y_i \qquad (3)$$

where δ is a vector of output values from a previous layer of processing, and the S's are previously calculated constants, one for each processor. In illustrating the pipeline processor in operation the output values of the various element for particular clock cycles are shown including the values on the memory address lines of the two memory banks. During weight update in the neural network back propogation, each weight is multiplied by a processor constant .

δ = 15, W = [100,50,40,73,17,43], Y = [35,69,152,13,42,237]

In the table, δ is fed from Stream A into register 1 (striped arrow) on clock cycle 2. On cycle 3, the first Y vector (35) is read from Stream A, into register 2 and a clock cycle 4, the contents of registers 1 and 2 are fed into the multiplier. Throughout each clock cycle stays in register 1. Also in cycle 4, the next Y vector (69) is read into register 2 and the first memory W vector value (100) is fed to register 0. On the next cycle the contents of the multiplier (6) are added to register 0 in the ALU (Adder) to give 106 to provide the first weight update calculation (106). This value is stored in register 4 on the sixth clock cycle and on the seventh cycle is written into its own memory on data back 0, i.e. the memory weight has been updated from 100 to 106. In the same way it will be seen that the procedure is repeated for each weight; thus W vector (50) is updated to (52) and (40) to (44) and so on.

In Fig. 18 the darker cells indicate reads from memory, whereas lighter shading indicates memory writes. The register values change in the same clock cycle as the associated computing element as a result of the 4 phase clock. X indicates undefined values.

The data processors 38 can be cascaded directly to increase the number of processors operating on the data stream. Fig 19 shows 3 processors 38a, b and c cascaded with the instruction processor being connected to 38a as described above. The connections made between the processors are depicted in Figs. 20a and 20b are a schematic illustration of a plurality of PVP units cascaded together to form a more powerful parallel vector processor neural network; the pin assignments for each pin are listed in the table shown in Fig. 21.

In a preferred configuration, the PVP calculation unit 30 contains 16 computing units each with 128 bytes of memory and with an operating frequency of 50 MHz. This allows the chip to perform calculations at the rate of 0.8 billion operations per second. The linear array architecture allows chips to be cascaded together if more computer units are required. Each chip will be about 8mm square and be packaged in a industry standard "Quad-Flatpack" surface mount chip carrier.

A realistic performance figure may be obtained by considering a practical application such as pattern recognition. In this case each processor performs convolutions using data flowing in one of the pipeline streams, with coefficients from local memory, allowing multiple features to be searched for simultaneously. The other pipeline stream is used as described above to output the results, sort them, using 2 of the remaining DMA engines with the larger the value obtained the closer the match. This provides an overall performance figure for a 16 processor parallel vector processing array of 0.8 billion calculations per second.

In the embodiment mentioned above, the PVP architecture is implemented in a circuit board connecting into the Sun Microsystem SPARCstation range of work-station computers via the internal S-bus as described above. The circuit board may be implemented in 2 versions, 1 version containing 4 PVP 32 chips and the other version containing 8 chips respectively, together with the interface circuitry, buffer memory and look up RAMs for the neural network algorithms. A schematic layout of a circuit board containing 8 PVP 32 chips is depicted in Fig. 22 of the drawings. This circuit board performs parallel comparison against 256 stored patterns on the valuation of a fully connected neural network with 2 layers, each containing 128 units with an overall calculation rate of 12.8 billion operations per second, compared to the basic Sun SPARCstation which calculates at the rate of 2 million operations per second or a Cray super computer which calculates at 2 billion operations per second.

The S-bus boards accelerate the core calculations and many pattern recognition in neural network applications and are particularly useful for data acquisition as performed off-line and data retrieval is from disk. The throughput of the board is comparable with the average transfer rate of data over the S-bus. Each board is supplied with software routines which allow users to access the power of the boards from their own programs while using the host computer for mass storage and display and overall control. In addition, the S-bus board also acts as a valuation platform for users who desire to assess the power of the PVP architecture when applied to the particular algorithms before committing to one of the more larger powerful designs.

It will also be understood that a more advanced system is required for users who require to process data as it is being produced and whose algorithms have more complex elements than just pattern matching or neural network algorithms calculated directly on the PVP chip. Some calculations performed by a host processor can easily become a bottleneck and reduce the throughput of PVP chips to much less than optimum and in many cases such bottleneck calculations are relatively straightforward data preparation and post-processing which can be speeded up dramatically by implementing them directly with relatively simple silicon integrated circuits.

Accordingly, the chips hereinbefore described and circuit boards can be implemented into a PVP desktop development system which is a modular system for connecting to a host bus, such as a Sun S-bus, for overall control upon which complete algorithms can be implemented in custom silicon and which can accept input directly from external cameras, microphones etc. and output to display devices and other data processing systems or process controllers.

Various modifications may be made to the embodiments hereinbefore described without departing from the scope of the invention. For example, any number of data processors may be conveniently cascaded and 4 or more data pipelines can be used to interconnect the data processors. More than one instruction processor could be used if required. The size and number of memories per chip can vary as can the size and number of the registers in the calculation unit. The stream co-processor could be augmented or replaced by other algorithm specific processors. Although the data processor uses an 8-bit word, it will be understood that other word sizes are useful, e.g. 16 or 32-bit words which increase performance. A pipeline is only one example of a communication network. A 2-dimensional array or other topology is possible and it is known that some algorithms benefit from other topologies. If each data processor has its own instruction processor the chip would then have Multiple Instruction, Multiple Data (MIMD) architecture instead of SIMD.

The PVP architecture can be used to accelerate calculations in a wide variety of applications particularly in the field of neural networks; graph matching; model-based image understanding; rank-order filtering where data is sorted into order and median value filtering which is used for noise reduction so that edges in images can be cleaned up. In particular, the PVP has applications in the fields of banking, insurance and legal, such as the automatic reading of cheques, invoices, signatures, contracts and in the Post Office to provide post code reading for sorting, in manufacturing for automatic inspection process control and computer-aided design, in the medical field, such as X-ray and ultrasonic image analysis and microscope sample analysis and in robotics for robotic vision processing, robot arm guidance and collision avoidance and also in communications for image compression and in video communications. The PVP chips can be used to accelerate neural network algorithms which are new adaptive classifications in any of these applications by using them in combination with a look-up RAM to post-process the output. The principal advantage of the invention is in the powerful performance, for example, a fully configured PVP desktop development system containing 64 PVP 32 chips will calculate at the rate of 128 billion operations per second. There is no existing computing system with this level of performance, nor is it envisaged that such a computing system could be offered for relatively inexpensive price of the present system which is a fraction of the cost of a Cray super computer.

CLAIMS

1. Parallel vector processing (PVP) apparatus for processing data, said parallel vector processing apparatus comprising,

at least one data processing means adapted to be coupled to at least one stream of data, said stream carrying data into and through said data processing means and carrying results from the data processing means after operations have been performed on the data, said data processing means having an internally addressable data memory, said internally addressable data memory having a plurality of stored coefficients for acting on the data in said stream, said data processing means including calculation means whereby operations are performed on the data in said stream by said coefficients retrieved from said data memory or on the data from the stream itself or on the coefficients themselves to provide a result which is passed to result stream and/or stored in the data memory, and an instruction processor means coupled to said at least one data processing means for supplying instructions to the data processing means to perform operations on data routed to said calculation means, the result of said operations being routed to said stream for output or stored in data memory.

2. Apparatus as claimed in claim 1 wherein said at least

one data processing means are coupled to at least two pipelined streams of data, one of said streams carrying data into and through said at least one data processing means and the other stream carrying results from said at least one data processing means.

3. Apparatus as claimed in claim 1 or claim 2 wherein the apparatus includes a plurality of data processing means each of which is coupled to each of said at least two pipelined data streams, and said instruction processor being coupled to one of data processing means.

4. Apparatus as claimed in claim 3 wherein the instructor processor is coupled to the first data processing means in the pipeline.

5. Apparatus as claimed in any one of claims 1-3 wherein several or all of said data processing means includes an instructor processor.

6. Apparatus as claimed in any preceding claim wherein the or each data processing means includes data routing means coupled between the data streams, the instruction processor and said internally addressable data memory, the arrangement being such that execution of said instructions is controlled by said data routing means whereby operations are performed on said data in said calculation means by said coefficients retrieved from said internally addressable data memory and the results of said operations are placed on said result stream or stored in data memory.

7. Apparatus as claimed in claim 6 wherein the data

routing means includes network switch means coupled to said data streams, data switch means, said calculation means and memory arbitration means.

8. Apparatus as claimed in any preceding claim wherein also a parallel vector processing chip is provided having a plurality of said data processing means and an instruction processor formed on a single silicon die which uses serial single instruction-multiple data (SIMD) architecture.

9. Apparatus as claimed in any preceding claim wherein each data processing unit includes at least one DMA engine and controller for transferring blocks of data to and from the data memory in each data processing units.

10. Apparatus as claimed in any preceding claim wherein multiple banks or memories are provided to allow simultaneous read/write of data to and from memory.

11. Apparatus as claimed in any preceding claim wherein the PVP is implemented in a system accelerator for interfacing with a bus-interface of existing workstations for increasing the speed of processing of said workstation, said accelerator having a PVP consisting of a plurality of such PVP chips, a data input port and a data output port, buffer memory means coupled to said input port and said output port, and data preparation means and post-processing means coupled between the PVP means and said buffer memory and an internal interface also coupled to the buffer memory interfacing with the bus-interface of

the workstation.

12. Parallel vector processing (PVP) apparatus for processing data in parallel, said PVP apparatus comprising:

a plurality of data processing means adapted to be coupled to at least two pipelined streams of data, one stream being coupled to each data processing means for carrying data to each of said data processing means, the other stream being coupled to each data processing means for carrying results from said data processing means after operations have been performed on the data, each of said data processing means having an internally addressable data memory having a plurality of coefficients stored therein for acting on said data in said one data stream, said data processing means including calculation means for performing operations on said stream by said coefficients retrieved from said data memory or on the data from the stream itself or on the coefficients themselves to provide a result on said other data stream and/or stored in the said data memory,

instruction processor means coupled to one of said data processing means for supplying instructions to all of said data processing means to perform operations on data routed to the calculation means,

the or each data processing means including data routing means coupled between said at least two data streams, said instruction processor and said internally addressable data memory, the arrangement being such that execution of said instructions is controlled by said data routing means whereby in each data processing means, coefficients are retrieved from said internally addressable memory and operate on data from said one data stream to provide a result, said result being passed to said other data stream or stored in data memory, said data processing means performing said operations simultaneously whereby input data is processed in parallel in said PVP apparatus.

13. Apparatus as claimed in claim 12 wherein the apparatus is conveniently implemented on a single silicon die.

14. A method of processing data, said method comprising the steps of:

storing a plurality of coefficients in an internally addressable data memory of at least one data processing means to operate on a stream of input data,

receiving said stream of input data from said at least one data pipeline and performing operations on said data or on the data from said stream itself or on the coefficients themselves by said coefficients from said data memory in said data processing means in accordance with instructions from instruction processing means coupled to said at least one data processing means to provide a result, routing said result to said stream for output or storing the result in data memory; and

repeating the procedure for a plurality of data elements in said data stream.

Conveniently, data is input on one data stream and the results are output on a second data stream.

15. A method as claimed in claim 14 wherein data is input on one data stream and the results are output on a second data stream.

16. A method as claimed in claim 15 wherein a plurality of data processing means are cascaded together and are coupled to said at least two streams to form a pipelined processor, said instruction processor being coupled to one of said data processing means, whereby the method outlined above is repeated in each data processing means for each element in the input data stream, so that simultaneous operations are performed on a plurality of different input data elements using coefficients from respective data processor memories, the results of said simultaneous operations being output onto said second pipelined data stream or stored in data memory.

17. A method as claimed in any one of claims 1-14 wherein the method is used to provide vector scalar product calculations for use in image processing, weight updating calculations for use in neural network applications or wavelet transformations for image compression.

18. A method of processing data in parallel to achieve faster processing of input data, said method combining the steps of,

providing a plurality of data processing means and coupling at least two pipelined streams of data to each of said data processing means, each data processor having an internally addressable data memory, storing a plurality of vector coefficients in each data memory,

passing data along one of said pipelined streams of data so that all data in said stream passes through each data processing means,

retrieving stored coefficients from said respective data memory under the control of instructions from an instruction processor coupled to one of said data processing means and operating on said data in said data processing means with said coefficients to provide a result,

outputting the result of each operation of said coefficients on said data onto said other of said data pipelines,

said operations in a plurality of said data processing means being carried out at the same time whereby the serial inputted pipelined data is processed in parallel.

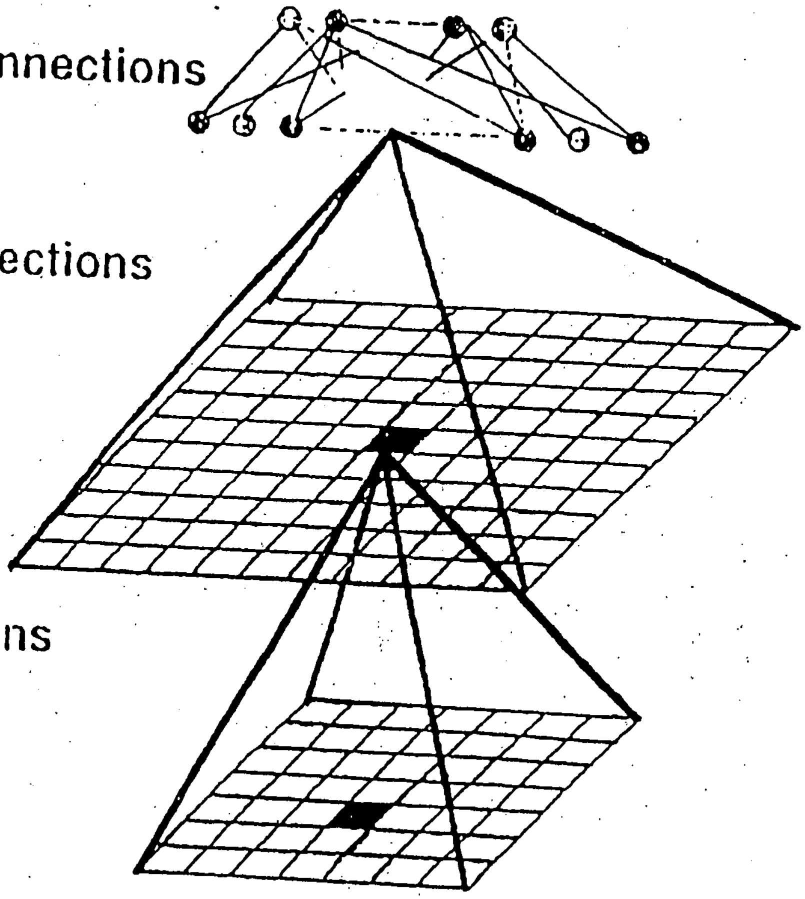

Object identification - 10 units
1000 connections
Feature extraction - 50 units
6000 connections
Noise reduction - 1 unit
49 connections

FIG.1

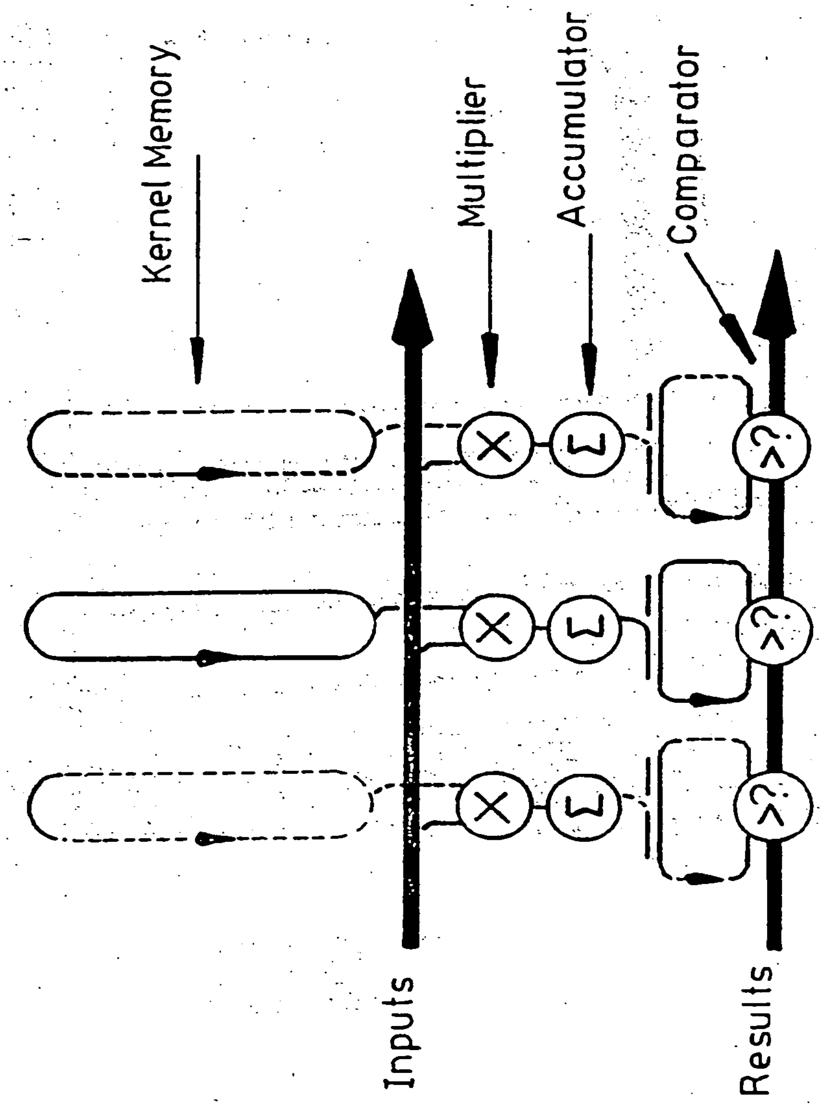

Kernel Memory
Multiplier
Accumulator
Comparator
Inputs
Results
×
Σ
>?

FIG. 2

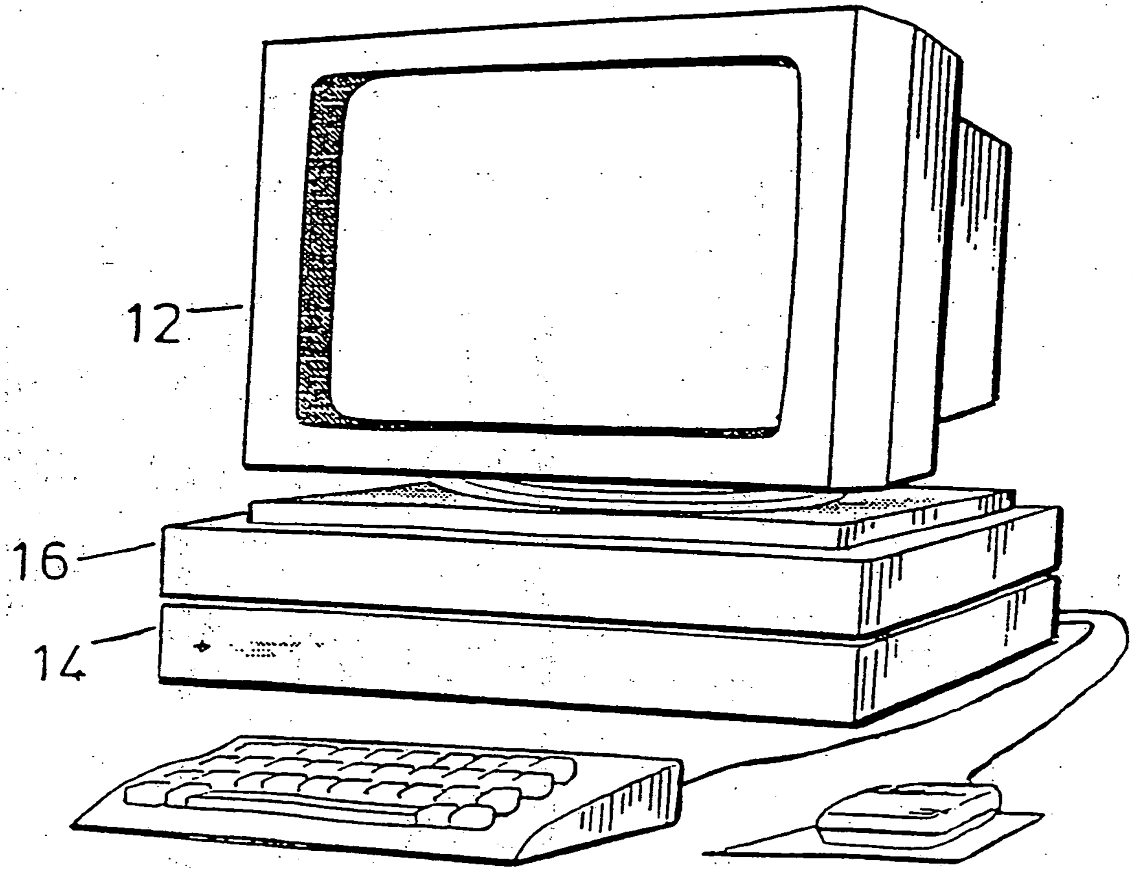


FIG. 3

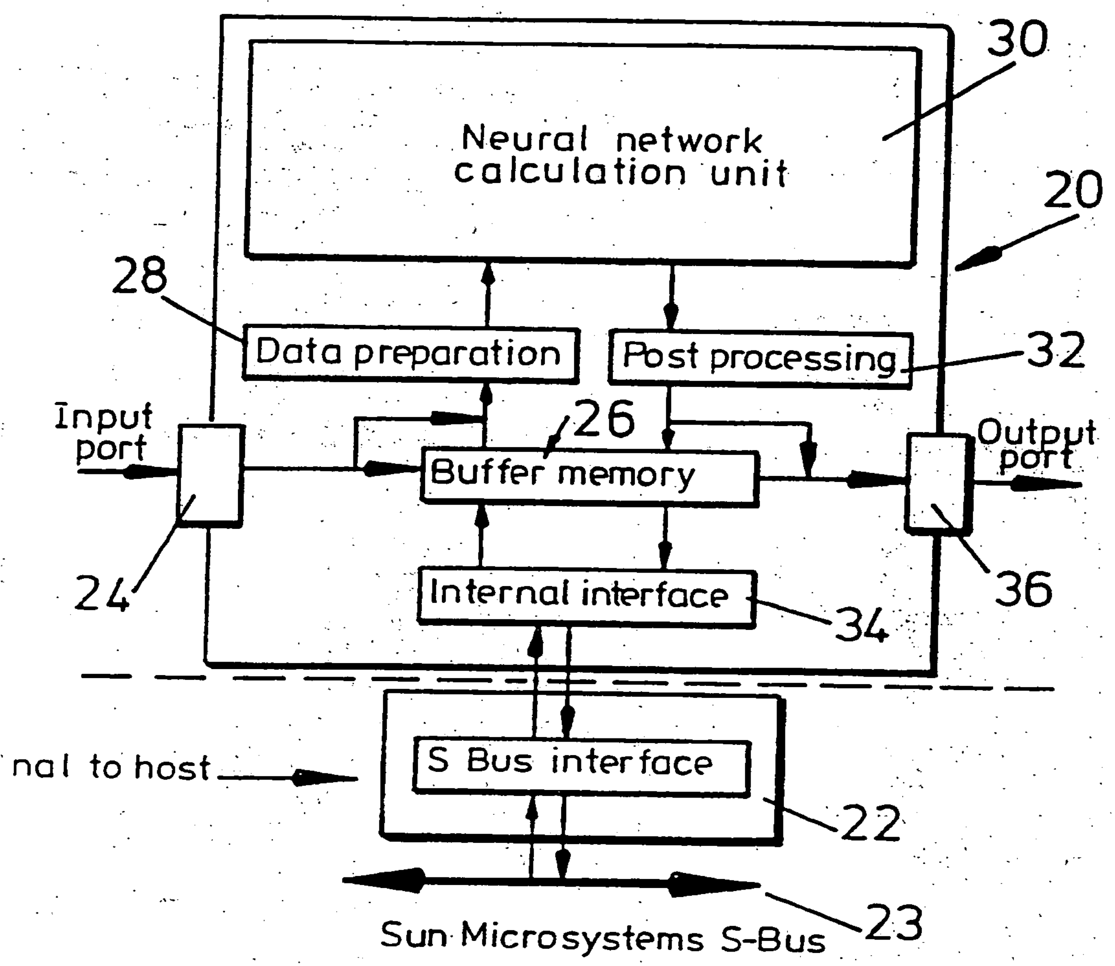

Neural network calculation unit
30
20
28
Data preparation
Post processing
32
Input port
26
Buffer memory
Output port
24
36
Internal interface
34
nal to host
S Bus interface
22
23
Sun Microsystems S-Bus

FIG.4

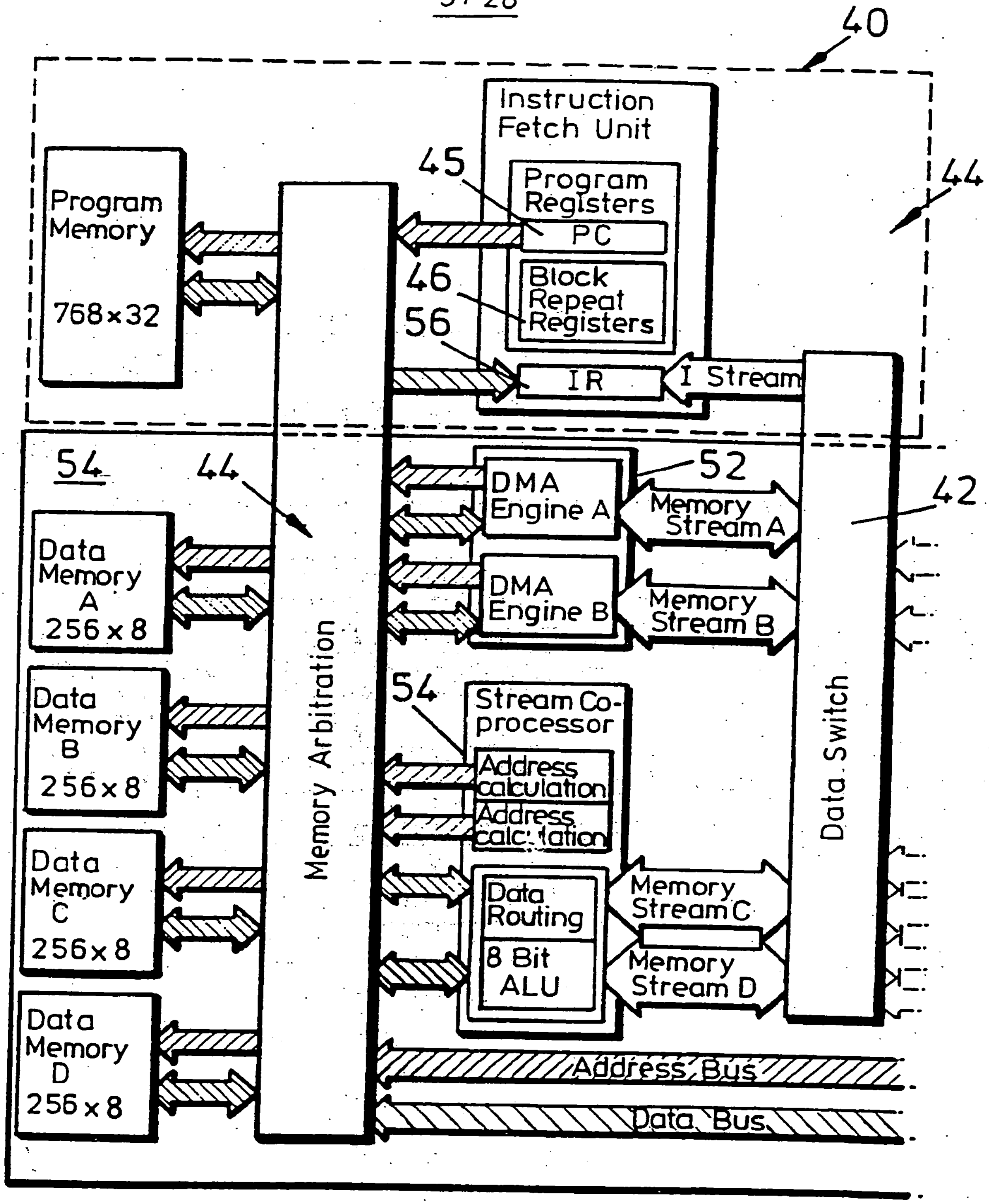

40
Instruction Fetch Unit
45
Program Registers
PC
44
Program Memory
768×32
46
Block Repeat Registers
56
IR
I Stream
54
44
DMA Engine A
52
Memory Stream A
42
DMA Engine B
Memory Stream B
Data Memory A 256×8
Data Memory B 256×8
Memory Arbitration
54
Stream Co-processor
Address calculation
Address calculation
Data.Switch
Data Memory C 256×8
Data Routing
8 Bit ALU
Memory Stream C
Memory Stream D
Data Memory D 256×8
Address Bus
Data Bus

FIG. 5A

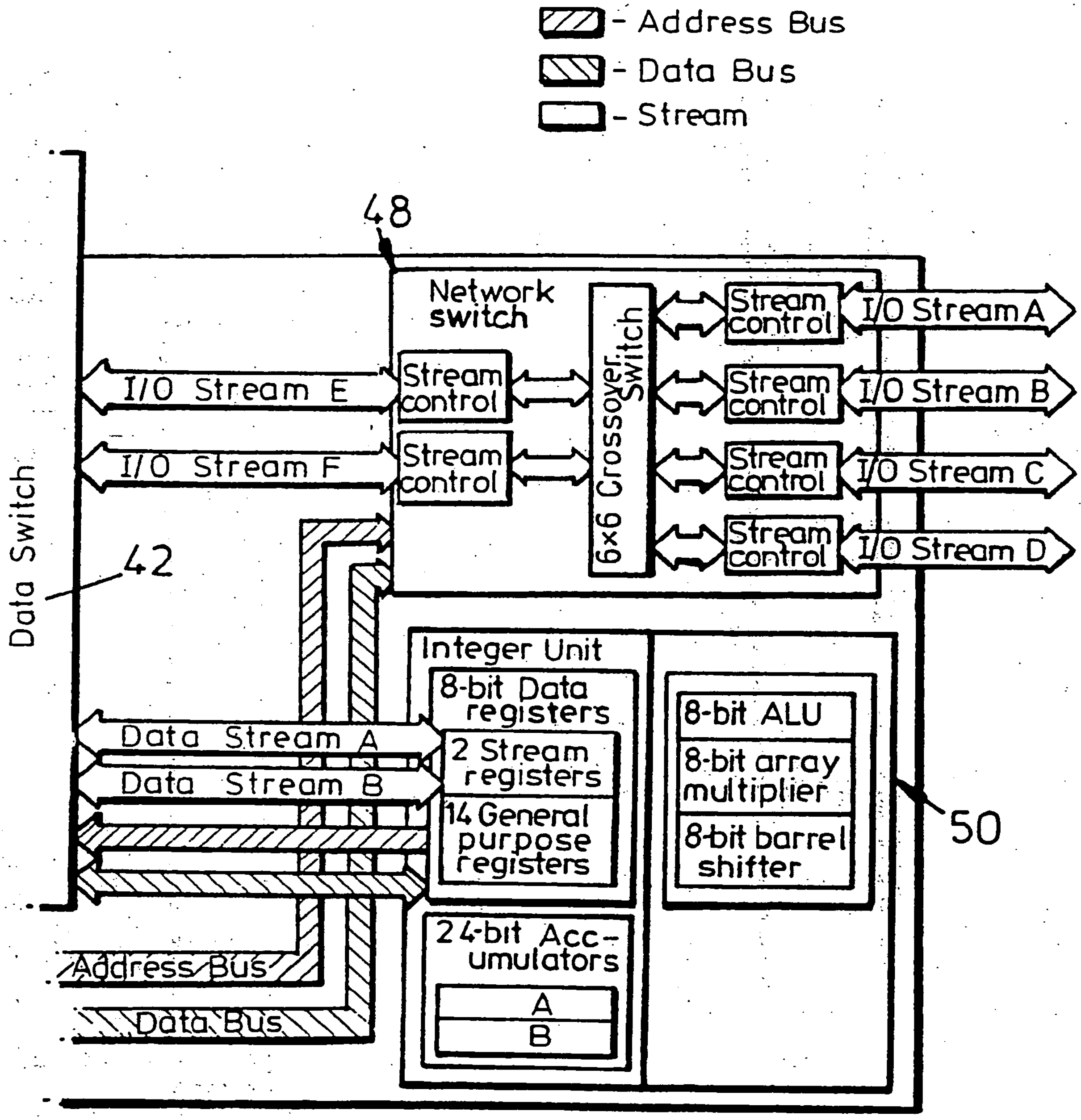

▨ - Address Bus
▨ - Data Bus
□ - Stream
48
Network switch
6x6 Crossover Switch
Stream control
Stream control
Stream control
Stream control
Stream control
Stream control
I/O Stream A
I/O Stream B
I/O Stream C
I/O Stream D
I/O Stream E
I/O Stream F
Data Switch
42
Data Stream A
Data Stream B
Address Bus
Data Bus
Integer Unit
8-bit Data registers
2 Stream registers
14 General purpose registers
24-bit Accumulators
A
B
8-bit ALU
8-bit array multiplier
8-bit barrel shifter
50

FIG.5B

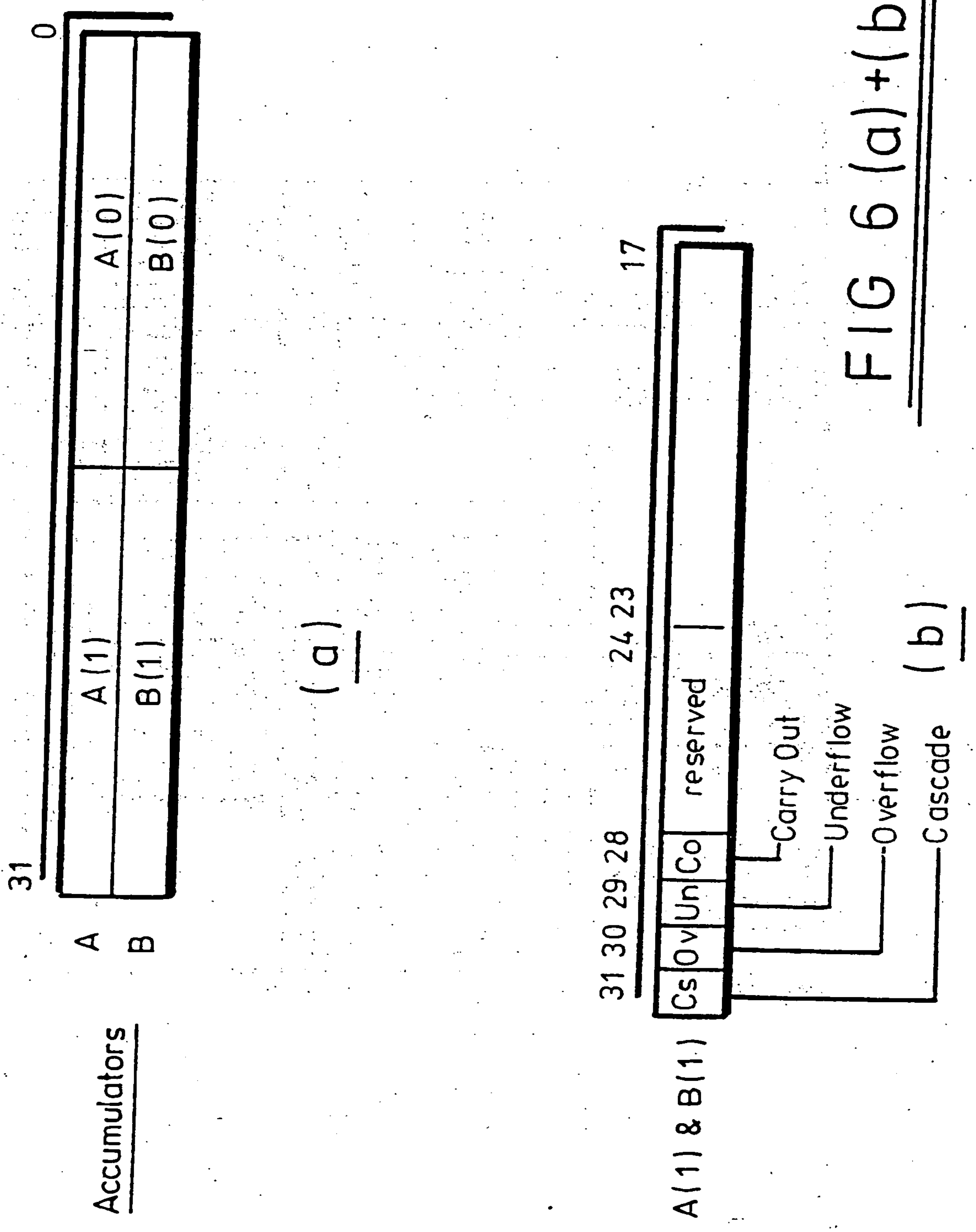

Accumulators
A
B
31
0
A(1)
A(0)
B(1)
B(0)
(a)
A(1) & B(1)
31 30 29 28
24 23
17
Cs
Ov
Un
Co
reserved
Carry Out
Underflow
Overflow
Cascade
(b)

FIG 6 (a)+(b)

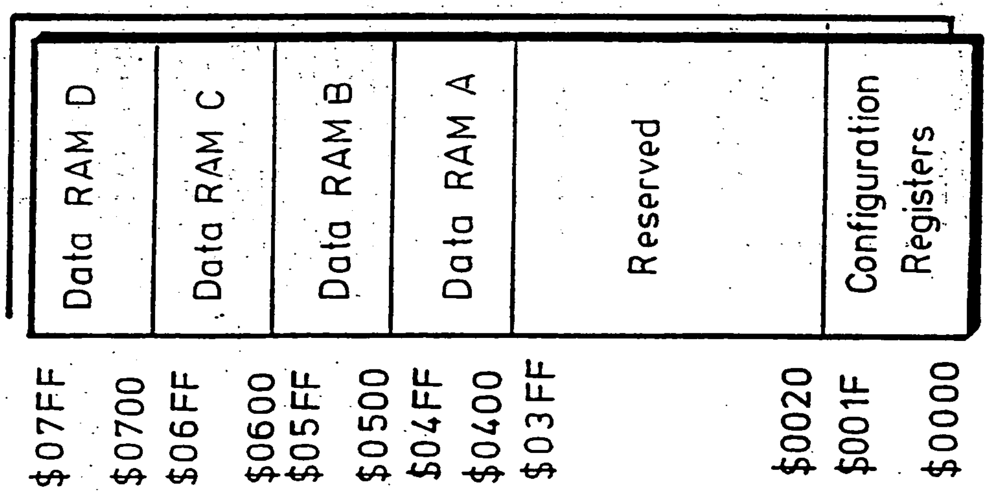

Data RAM D
Data RAM C
Data RAM B
Data RAM A
Reserved
Configuration Registers
$07FF
$0700
$06FF
$0600
$05FF
$0500
$04FF
$0400
$03FF
$0020
$001F
$0000

FIG.7

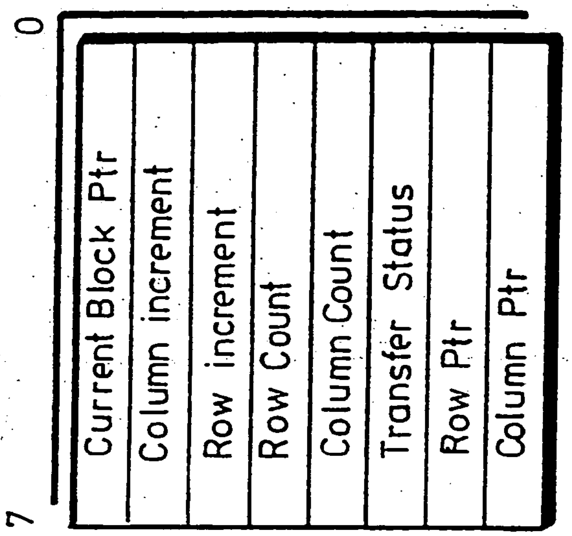

0
7
Current Block Ptr
Column increment
Row increment
Row Count
Column Count
Transfer Status
Row Ptr
Column Ptr

FIG.9

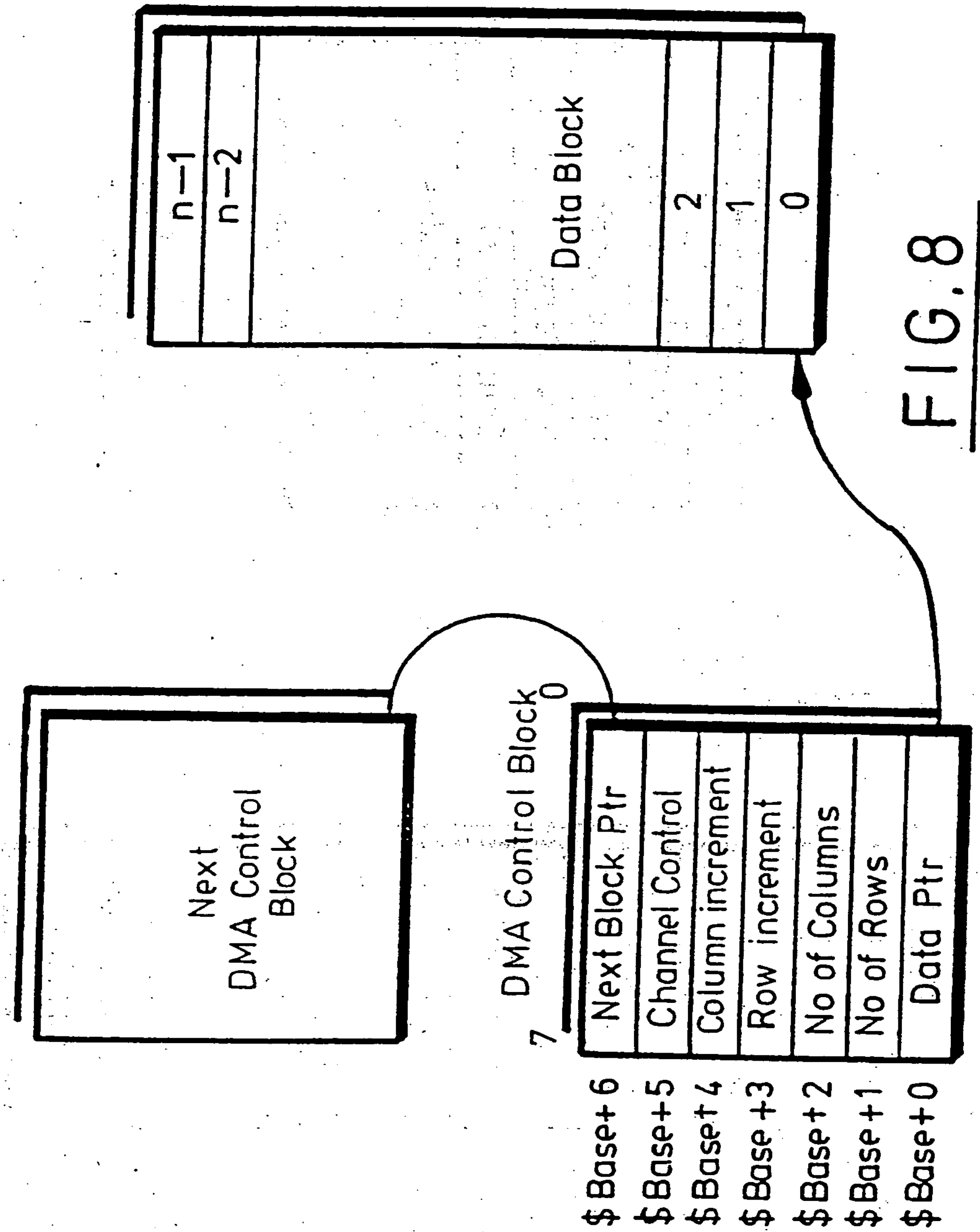

Data Block
n−1
n−2
2
1
0
Next
DMA Control
Block
DMA Control Block
0
7
Next Block Ptr
Channel Control
Column increment
Row increment
No of Columns
No of Rows
Data Ptr
$Base+6
$Base+5
$Base+4
$Base+3
$Base+2
$Base+1
$Base+0

FIG.8

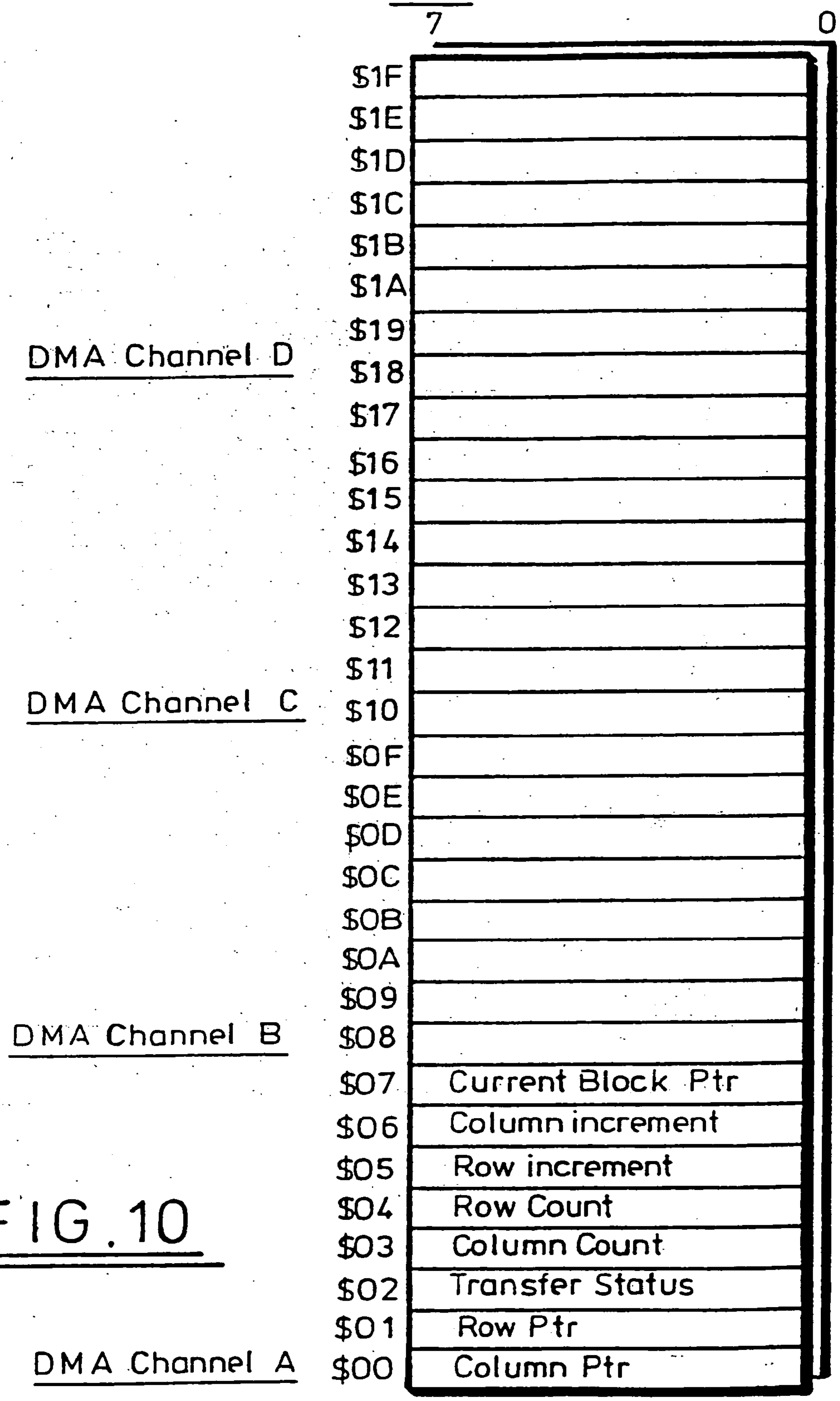

7
0
$1F
$1E
$1D
$1C
$1B
$1A
$19
DMA Channel D
$18
$17
$16
$15
$14
$13
$12
$11
DMA Channel C
$10
$0F
$0E
$0D
$0C
$0B
$0A
$09
DMA Channel B
$08
$07 Current Block Ptr
$06 Column increment
$05 Row increment
$04 Row Count
$03 Column Count
$02 Transfer Status
$01 Row Ptr
DMA Channel A
$00 Column Ptr

FIG.10

Transfer	Ld	Load half-word from memory
	St	Store half-word to memory
	Ldb	Load byte from memory
	Stb	Store byte to memory
	Mov	register to register move
Arithmetic	Add	2's complement addition
	Addu	unsigned addition
	Sub	2's complement subtraction
	Subu	unsigned subtraction
	Mul	2's complement multiplication
	Mulu	unsigned multiplication
	Mac	multiply-accumulate
	Macu	unsigned multiply-accumulate
	Sxb	Sign Extend byte to half-word
Logical	And	bit-wise logical AND
	Or	bit-wise logical OR
	Xor	bitwise logical Exclusive-OR
Shift	Lsl	logical shift left
	Lsr	logical shift right
	Asl	arithmetic shift left
	Asr	arithmetic shift right
	Rol	rotate left
	Ror	rotate right
Compare	Ord	order byte pair
Branch		

FIG. 11

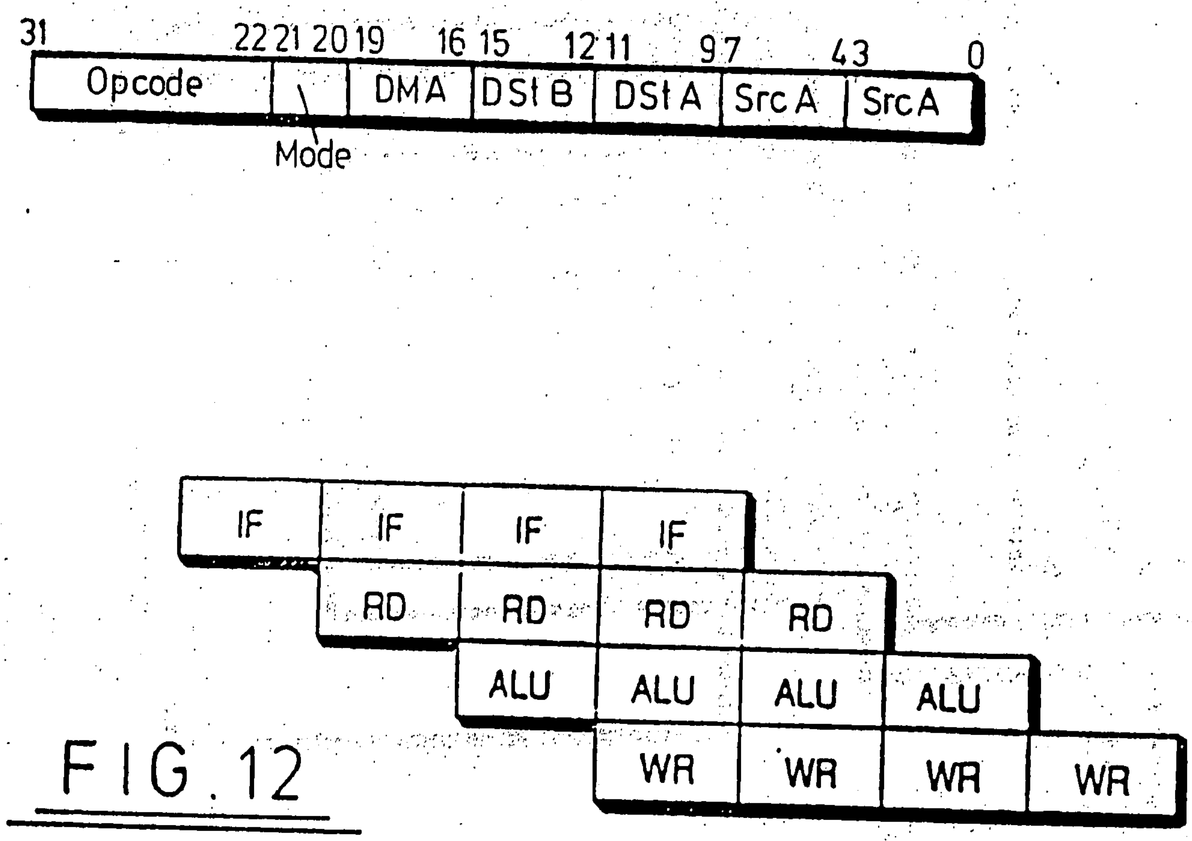

31
22
21
20
19
16
15
12
11
9
7
4
3
0
Opcode
Mode
DMA
DSt B
DSt A
Src A
Src A
IF
IF
IF
IF
RD
RD
RD
RD
ALU
ALU
ALU
ALU
WR
WR
WR
WR

FIG. 12

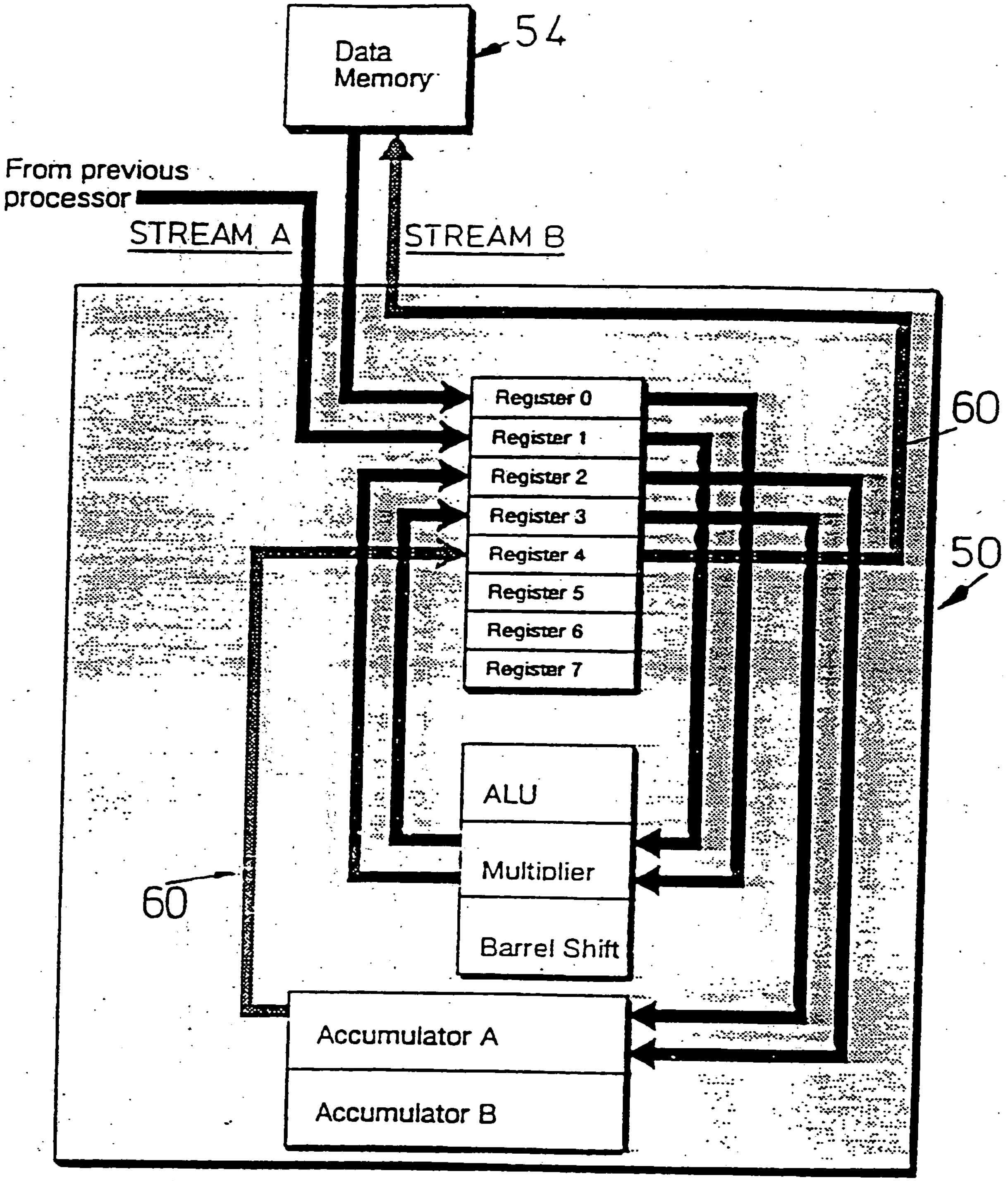

Data Memory
54
From previous processor
STREAM A
STREAM B
Register 0
Register 1
Register 2
Register 3
Register 4
Register 5
Register 6
Register 7
60
50
ALU
Multiplier
Barrel Shift
60
Accumulator A
Accumulator B

FIG. 13

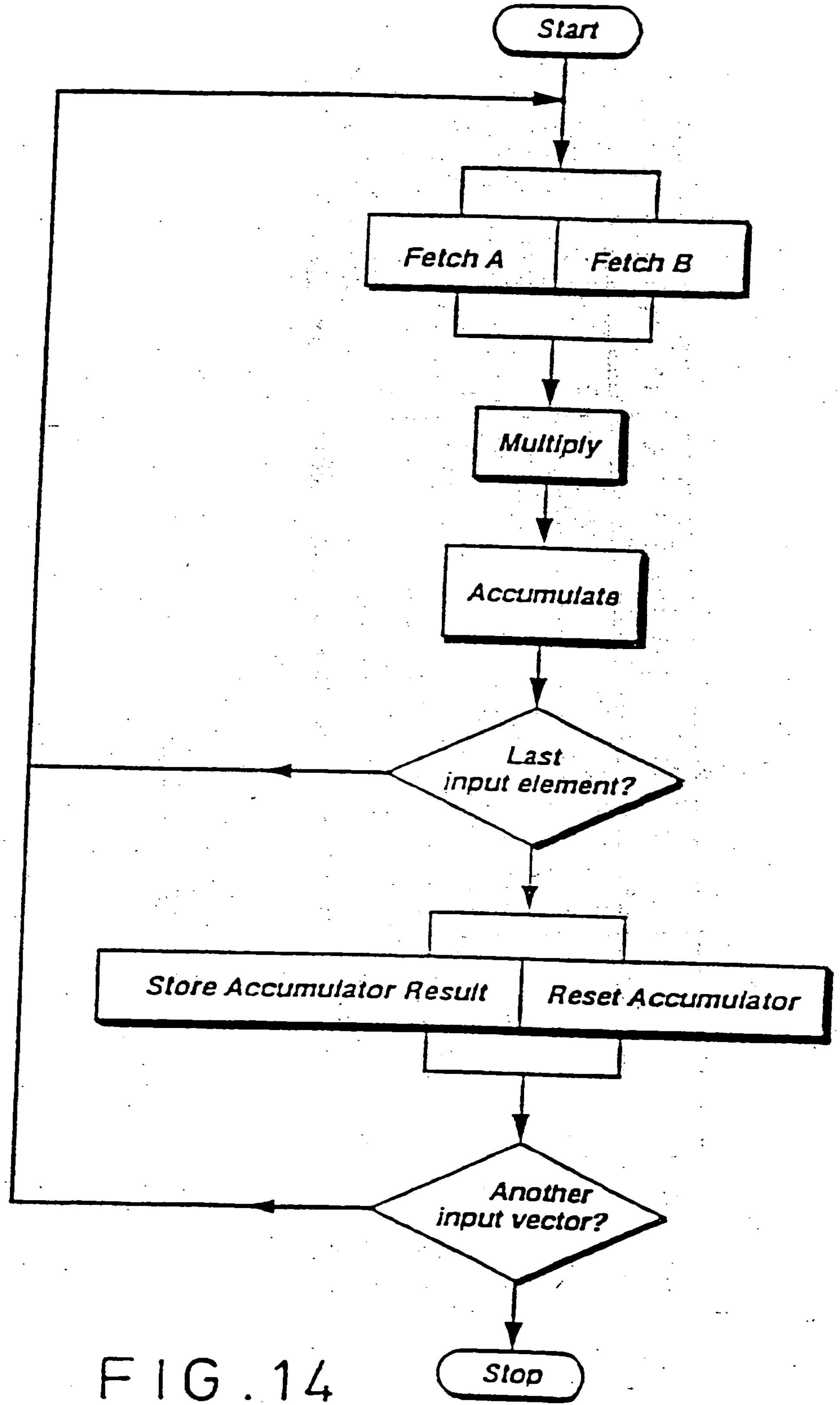

Start
Fetch A
Fetch B
Multiply
Accumulate
Last input element?
Store Accumulator Result
Reset Accumulator
Another input vector?
Stop

FIG. 14

Clock cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Stream A	1	2	3	X	2	1	5	X	4	3	1	X	X	X
Fetched from memory	4	5	6	X	4	5	6	X	4	5	6	X	X	X
Multiplier	X	4	10	8	X	8	5	30	X	16	15	6	X	X
Accumulator A	X	X	4	14	32	X	8	13	43	X	16	31	37	X
Stream D coprocessor	X	X	X	X	X	32	X	X	X	43	X	X	X	37

FIG.15

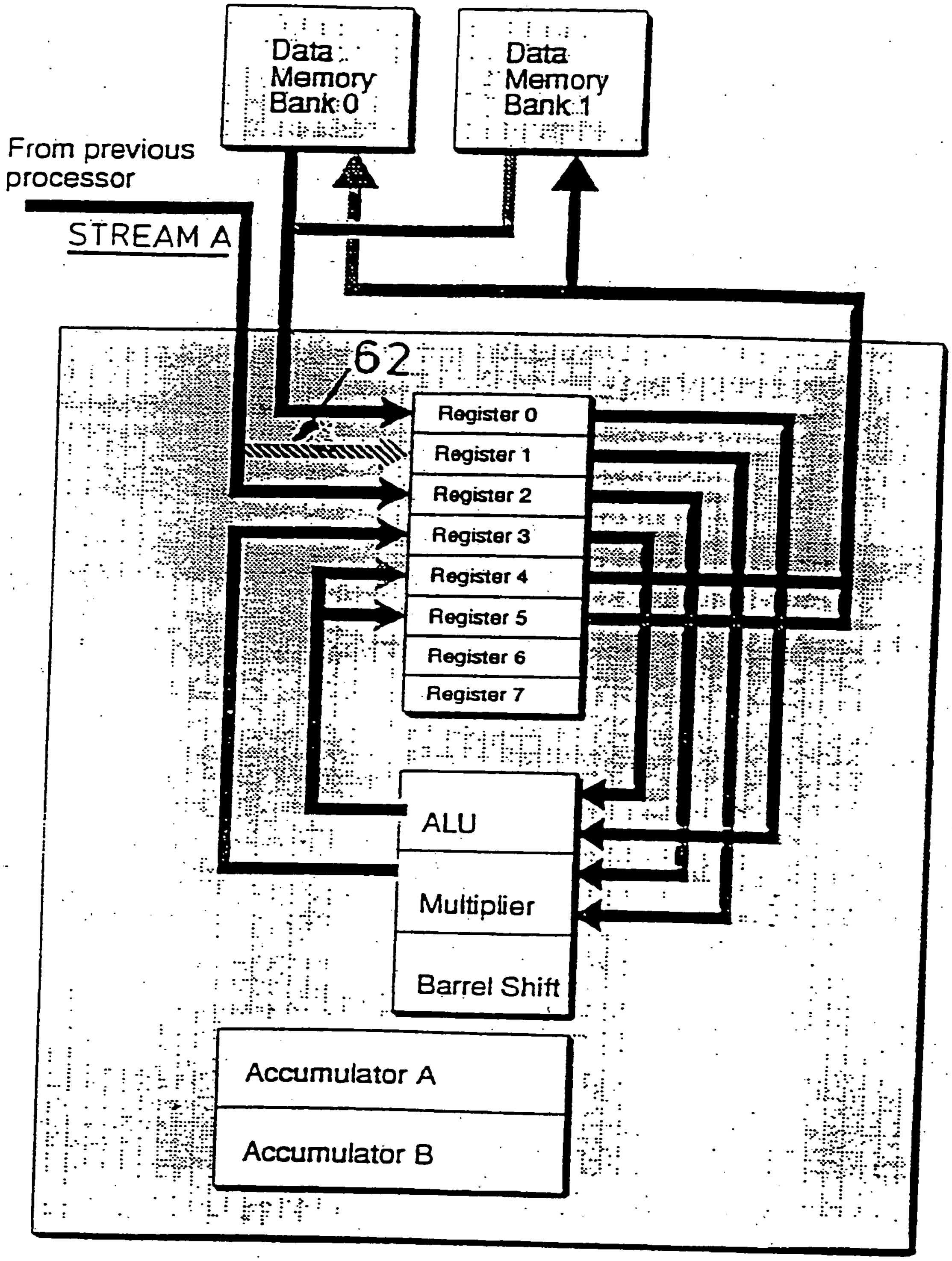

Data Memory Bank 0
Data Memory Bank 1
From previous processor
STREAM A
62
Register 0
Register 1
Register 2
Register 3
Register 4
Register 5
Register 6
Register 7
ALU
Multiplier
Barrel Shift
Accumulator A
Accumulator B

FIG.16

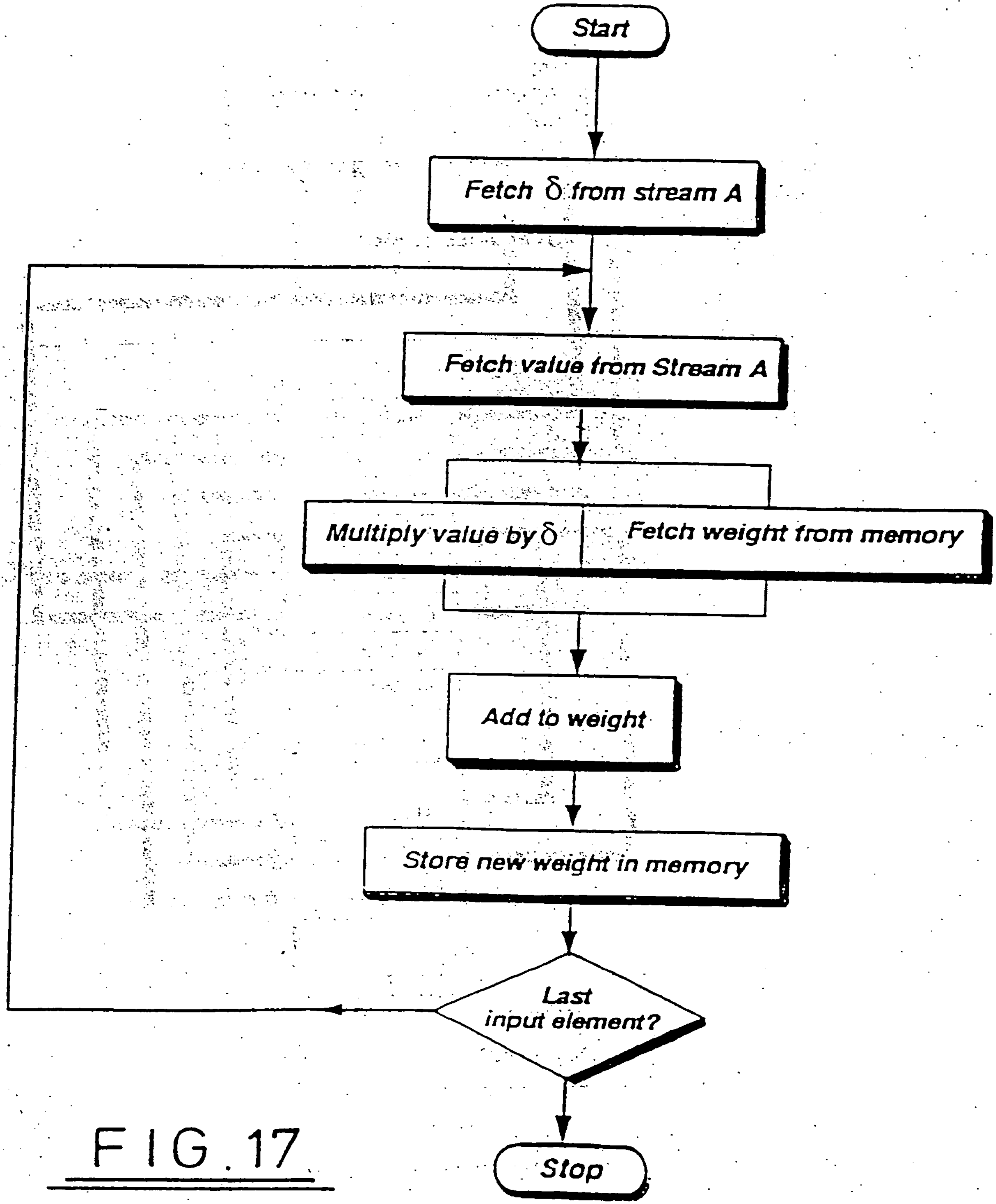

Start
Fetch δ from stream A
Fetch value from Stream A
Multiply value by δ
Fetch weight from memory
Add to weight
Store new weight in memory
Last input element?
Stop

FIG.17

Clock cycle	1	2	3	4	5	6	7	8	9	10	11	12
Stream A	X	15	35	69	152	13	42	237	X	X	X	X
Memory AddressBus0	X	X	X	0	X	[illegible]	0	2	1	X	2	X
Memory DataBus0	X	X	X	100	X	40	108	17	44	X	18	X
Memory AddressBus1	X	X	X	X	0	X	1	0	2	1	X	2
Memory DataBus1	X	X	X	X	50	X	73	52	45	82	X	47
Delta (register 1)	X	15	15	15	15	15	15	15	15	15	15	15
Multiplier (top byte)	X	X	X	6	2	4	9	1	2	14	X	X
ALU (Adder)	X	X	X	X	106	52	44	82	18	47	X	X
Register 4	X	X	X	X	106	106	44	44	18	18	18	X
Register 5	X	X	X	X	X	X	52	52	82	82	47	47

FIG.18

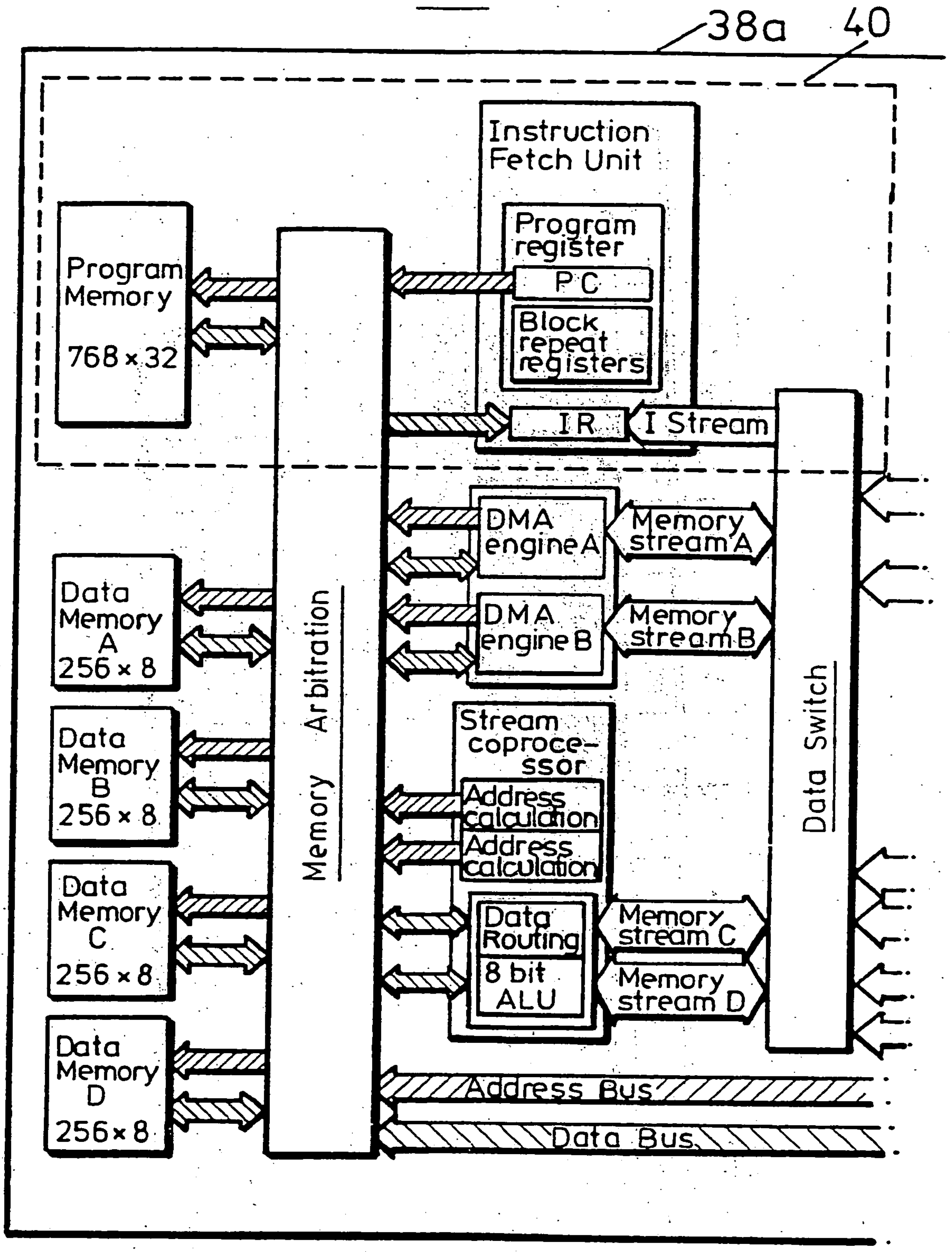

38a
40
Instruction Fetch Unit
Program register
PC
Block repeat registers
Program Memory
768×32
IR
I Stream
Memory Arbitration
Data Switch
DMA engine A
Memory stream A
DMA engine B
Memory stream B
Data Memory A 256×8
Data Memory B 256×8
Data Memory C 256×8
Data Memory D 256×8
Stream coprocessor
Address calculation
Address calculation
Data Routing
8 bit ALU
Memory stream C
Memory stream D
Address Bus
Data Bus

FIG.19A

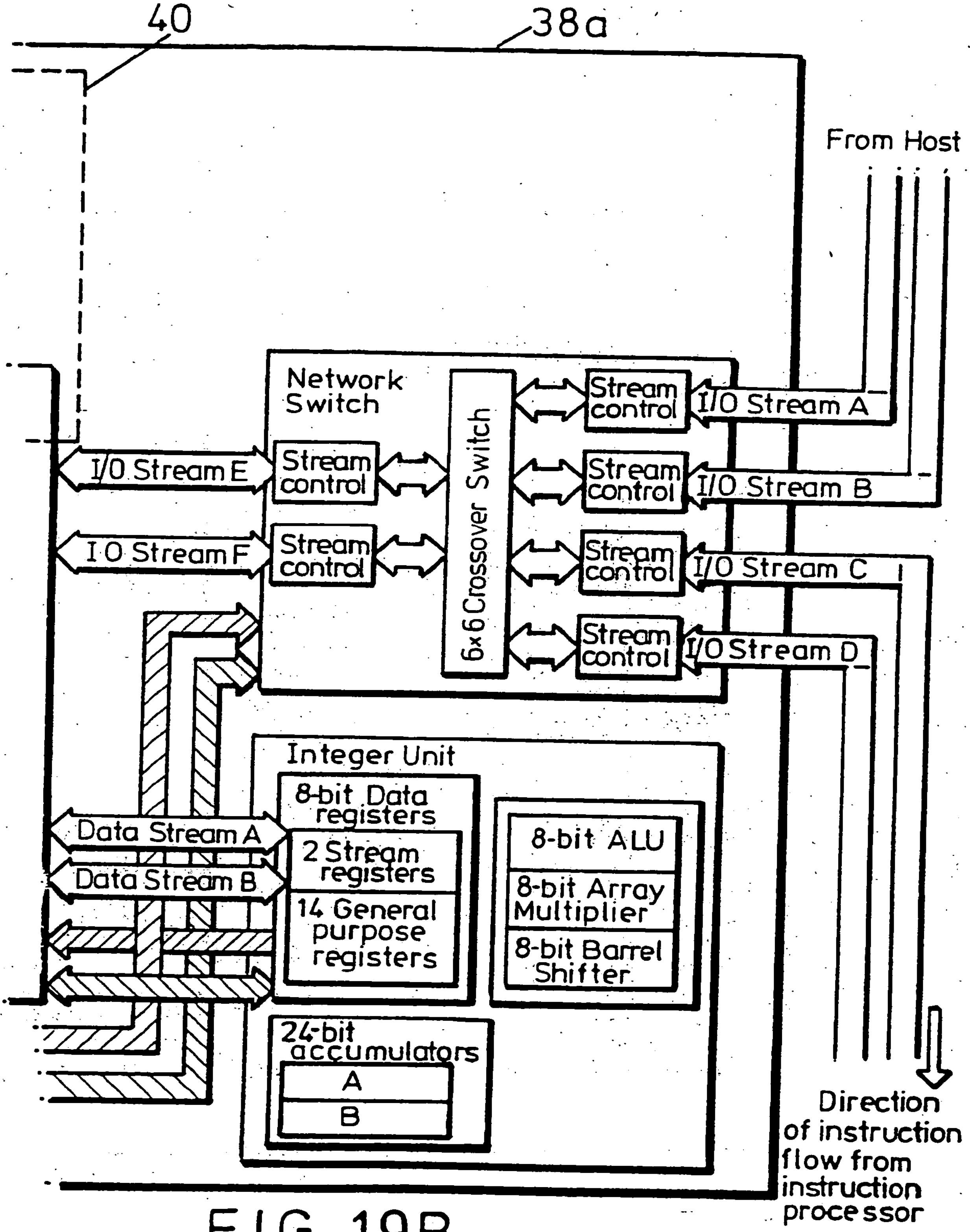

40
38a
From Host
Network Switch
6×6 Crossover Switch
Stream control
Stream control
Stream control
Stream control
Stream control
Stream control
I/O Stream A
I/O Stream B
I/O Stream C
I/O Stream D
I/O Stream E
I O Stream F
Integer Unit
8-bit Data registers
2 Stream registers
14 General purpose registers
8-bit ALU
8-bit Array Multiplier
8-bit Barrel Shifter
24-bit accumulators
A
B
Data Stream A
Data Stream B
Direction of instruction flow from instruction processor

FIG.19B

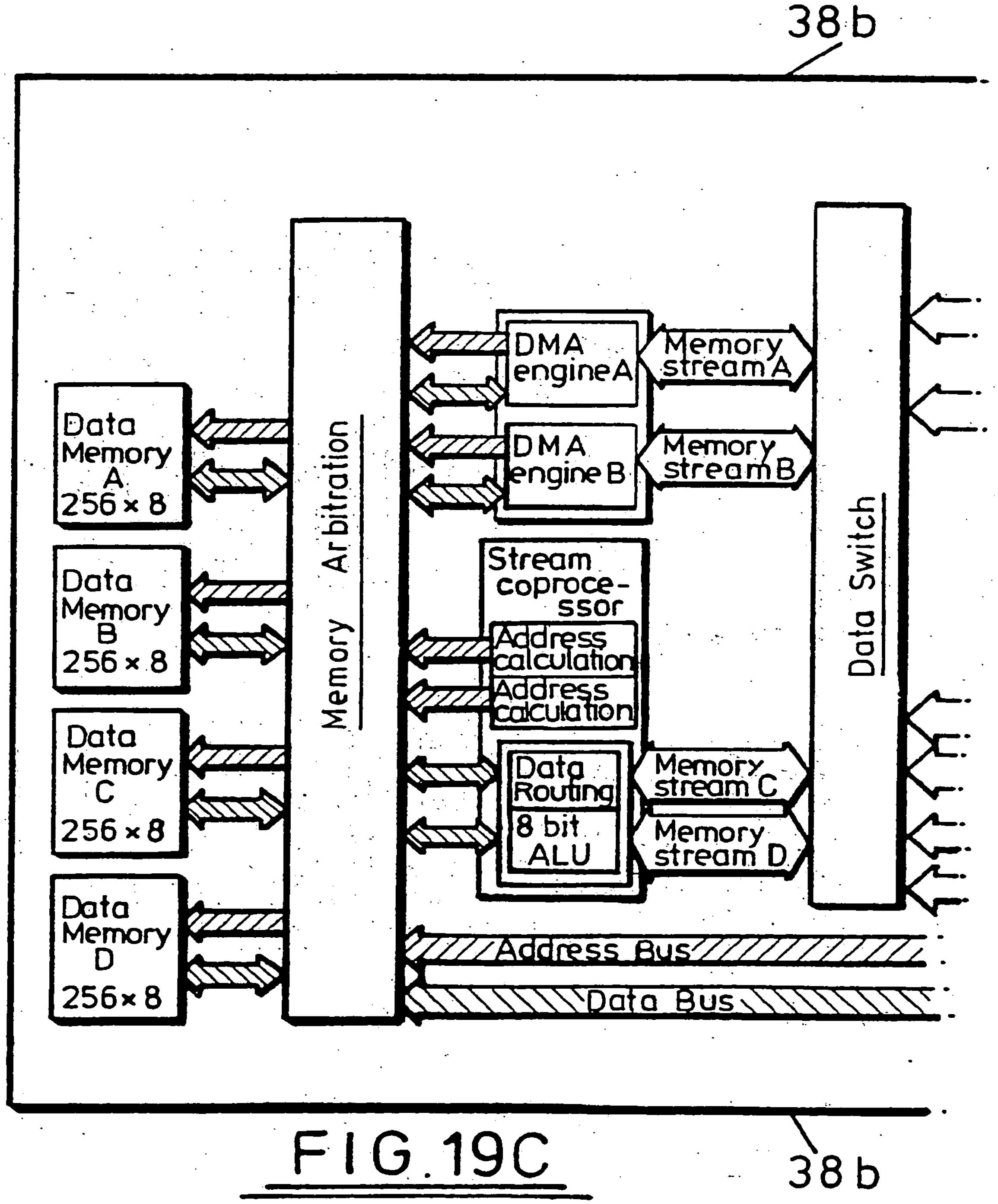

38b
Data Memory A 256×8
Data Memory B 256×8
Data Memory C 256×8
Data Memory D 256×8
Memory Arbitration
DMA engine A
DMA engine B
Memory stream A
Memory stream B
Stream coprocessor
Address calculation
Address calculation
Data Routing
8 bit ALU
Memory stream C
Memory stream D
Data Switch
Address Bus
Data Bus
38b

FIG.19C

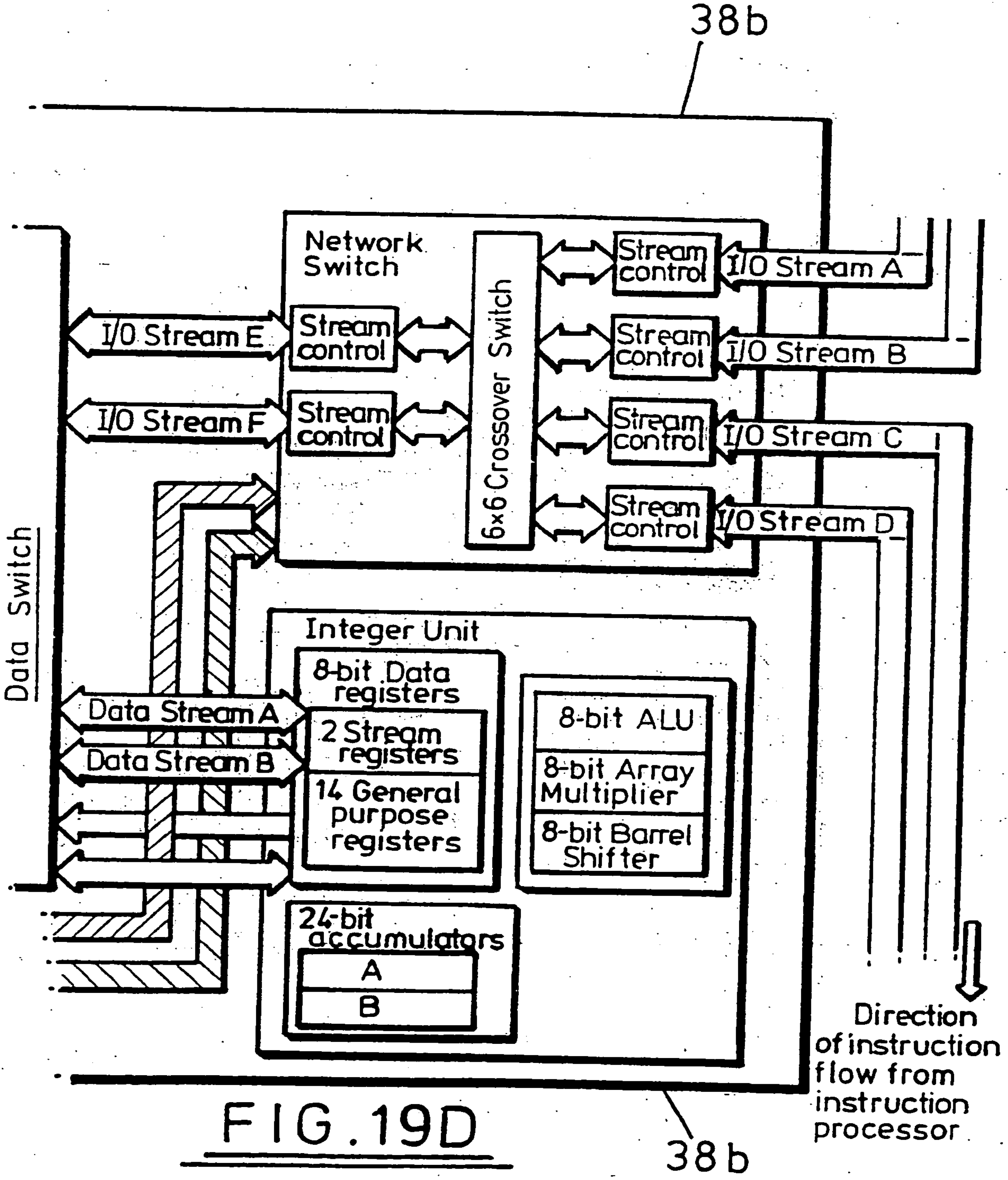

38b
Network Switch
6x6 Crossover Switch
Stream control
Stream control
Stream control
Stream control
Stream control
Stream control
I/O Stream A
I/O Stream B
I/O Stream C
I/O Stream D
I/O Stream E
I/O Stream F
Data Switch
Integer Unit
8-bit Data registers
2 Stream registers
14 General purpose registers
8-bit ALU
8-bit Array Multiplier
8-bit Barrel Shifter
24-bit accumulators
A
B
Data Stream A
Data Stream B
Direction of instruction flow from instruction processor
38b

FIG.19D

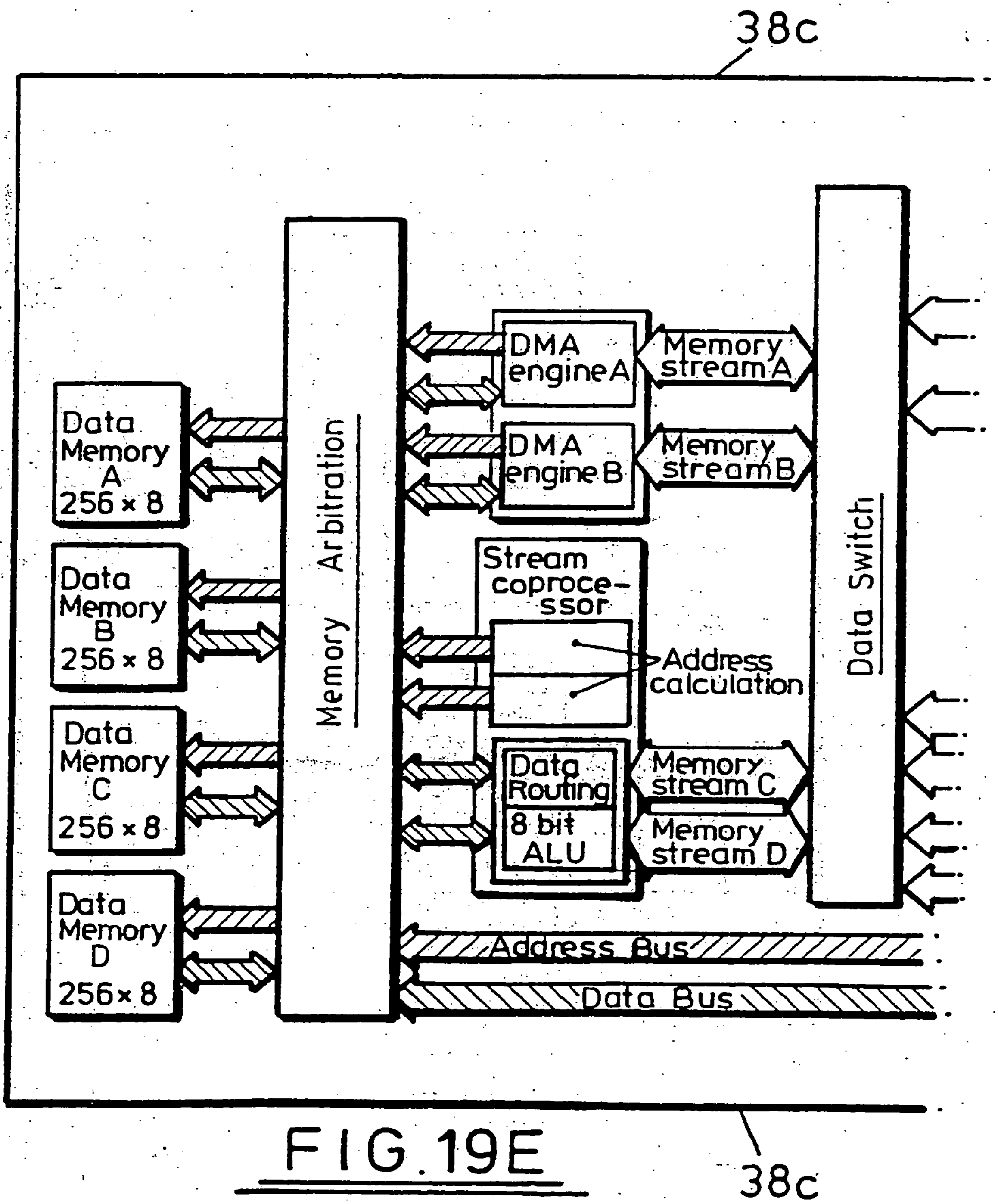

38c
Data Memory A 256 × 8
Data Memory B 256 × 8
Data Memory C 256 × 8
Data Memory D 256 × 8
Memory Arbitration
DMA engine A
DMA engine B
Memory stream A
Memory stream B
Stream coprocessor
Address calculation
Data Routing
8 bit ALU
Memory stream C
Memory stream D
Data Switch
Address Bus
Data Bus
38c

FIG. 19E

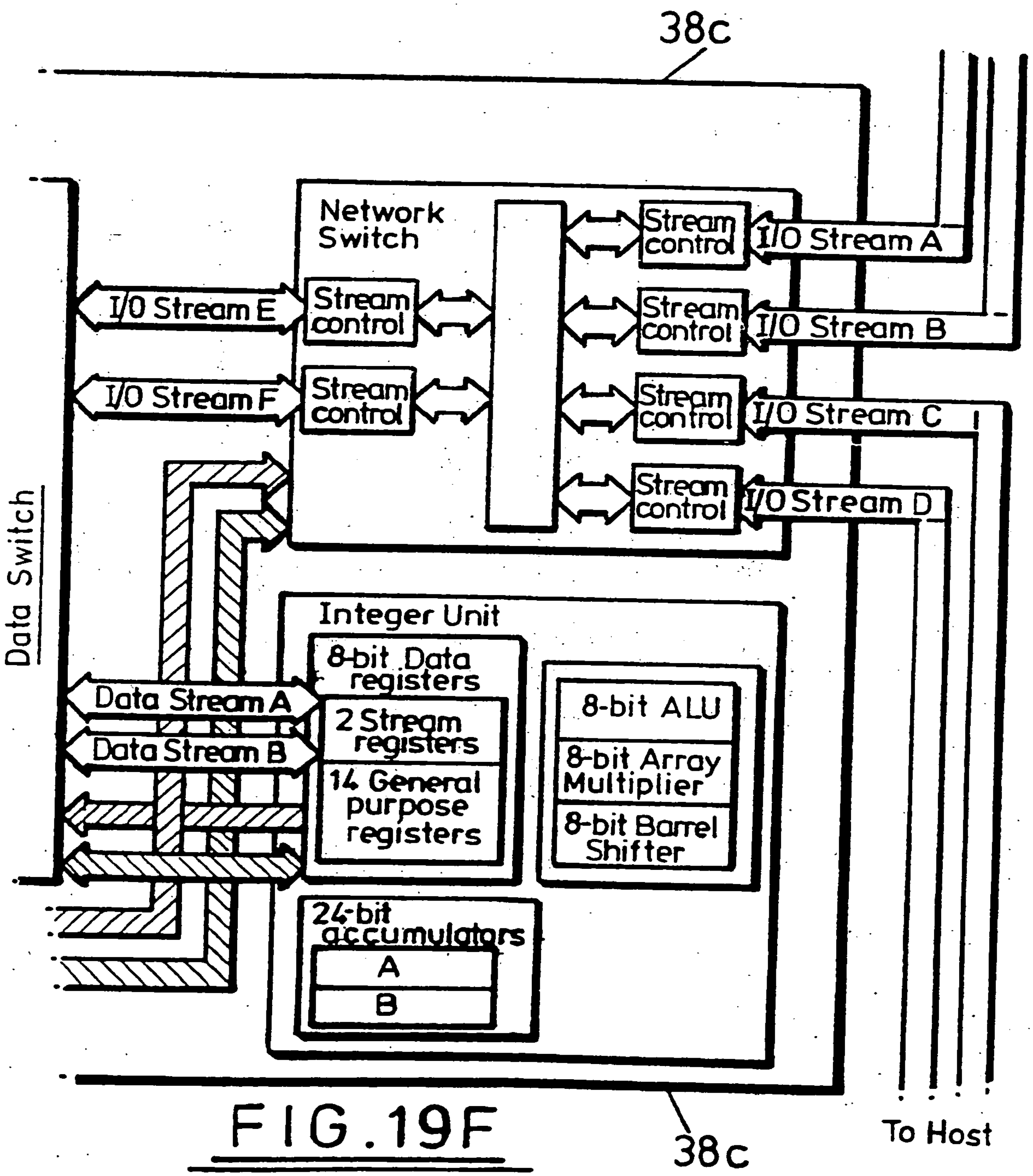

38c
Network Switch
Stream control
I/O Stream A
I/O Stream B
I/O Stream C
I/O Stream D
I/O Stream E
I/O Stream F
Data Switch
Integer Unit
8-bit Data registers
2 Stream registers
14 General purpose registers
8-bit ALU
8-bit Array Multiplier
8-bit Barrel Shifter
24-bit accumulators
A
B
Data Stream A
Data Stream B
38c
To Host

FIG.19F

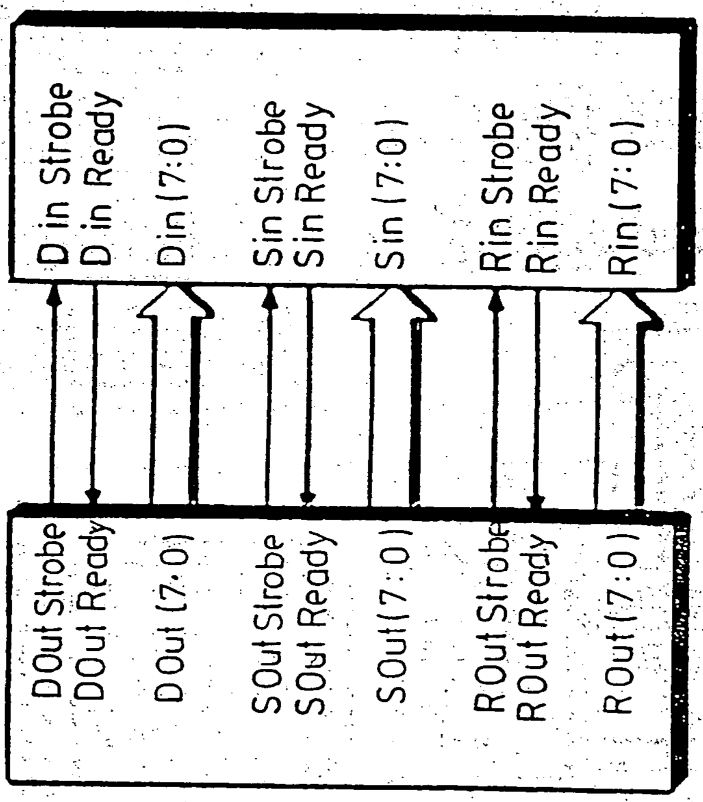

D in Strobe
D in Ready
Din (7:0)
Sin Strobe
Sin Ready
Sin (7:0)
Rin Strobe
Rin Ready
Rin (7:0)
DOut Strobe
DOut Ready
DOut (7:0)
SOut Strobe
SOut Ready
SOut (7:0)
ROut Strobe
ROut Ready
ROut (7:0)

FIG 20a

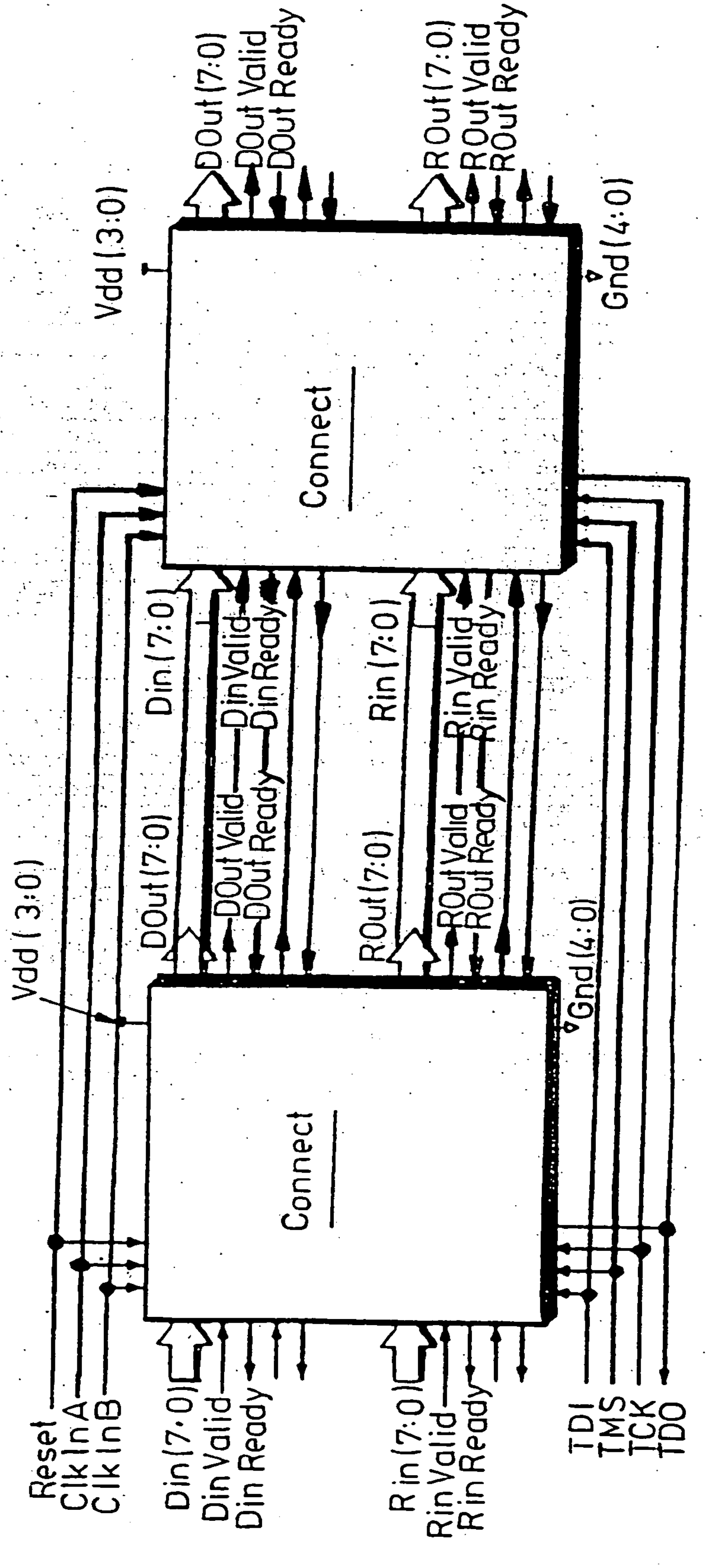

Vdd(3:0)
DOut(7:0)
DOut Valid
DOut Ready
ROut(7:0)
ROut Valid
ROut Ready
Gnd(4:0)
Connect
Din(7:0)
Din Valid
Din Ready
Rin(7:0)
Rin Valid
Rin Ready
DOut(7:0)
DOut Valid
DOut Ready
ROut(7:0)
ROut Valid
ROut Ready
Vdd(3:0)
Gnd(4:0)
Connect
Reset
Clk In A
Clk In B
Din(7:0)
Din Valid
Din Ready
R in(7:0)
Rin Valid
Rin Ready
TDI
TMS
TCK
TDO

FIG. 20b

Stream A	In	SAIn[7:0]	Stream A input data
		SAInVal	Stream A input data is valid
		SAInRdy	Stream A input is ready for more data
		SAInReq	Request Stream ownership from Connect
		SAInAck	Grant ownership of Stream
	Out	SAOut[7:0]	Stream A output data
		SAOutVal	Output data is valid
		SAOutRdy	Output is ready for more data
		SAOutReq	Connect requests Stream ownership
		SAOutAck	output grants Stream ownership
Stream B	In	SBIn[7:0]	Stream B input data
		SBInVal	Stream B input data is valid
		SBInRdy	Stream B input is ready for more data
		SBInReq	Request Stream ownership from Connect
		SBInAck	Grant ownership of Stream
	Out	SBOu[7:0]	Stream B output data
		SBOutVal	Output data is valid
		SBOutRdy	Output is ready for more data
		SBOutReq	Connect requests Stream ownership
		SBOutAck	output grants Stream ownership
Configuration		Config[2:0]	Configure port protocol
JTAG		TDI	Test Data In, serial data in
		TDO	Test Data Out, serial data out
		TMS	Test Mode Select, change JTAG state
		TCK	Test Clock to clock serial data
System Services		ClkA	Crystal or Oscillator input
		ClkB	Crystal input
		Reset	Power-on or system reset
Power Supply		Vdd[3:0]	Positive (+5V) voltage supply
cline3-4		Gnd[5:0]	Power supply ground (0V)
hline			

FIG. 21

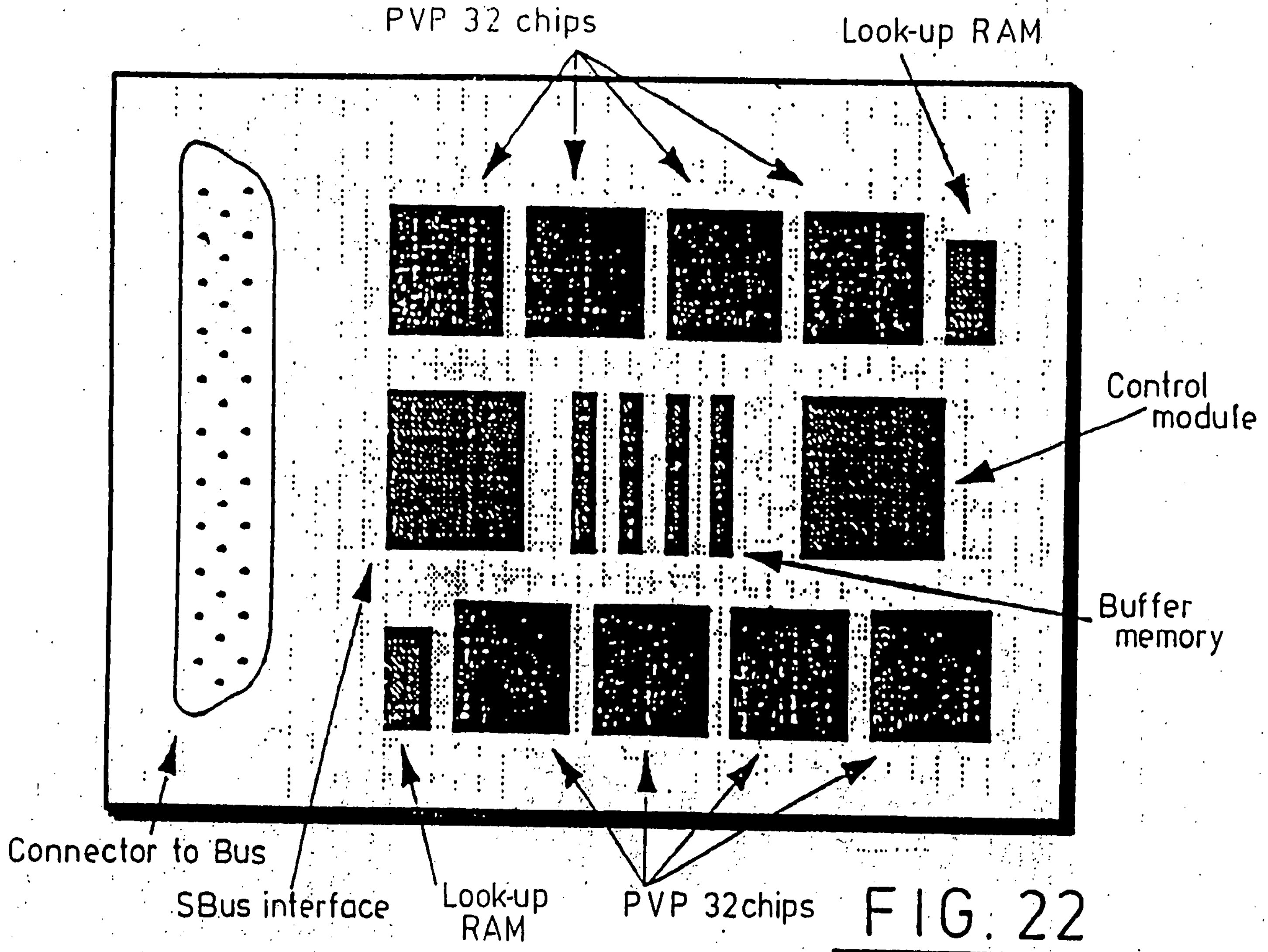

PVP 32 chips
Look-up RAM
Control module
Buffer memory
Connector to Bus
SBus interface
Look-up RAM
PVP 32chips

FIG. 22

INTERNATIONAL SEARCH REPORT

International Application No PCT/GB 93/00580

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all)[6]

According to International Patent Classification (IPC) or to both National Classification and IPC

Int.Cl. 5 G06F15/80; G06F15/66

II. FIELDS SEARCHED

Minimum Documentation Searched[7]

Classification System	Classification Symbols
Int.Cl. 5	G06F

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched[8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT[9]

Category °	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No.[13]
X	IEE PROCEEDINGS E. COMPUTERS & DIGITAL TECHNIQUES vol. 135, no. 4, July 1988, STEVENAGE GB pages 214 - 221 F. WOODHAMS AND W. PRICE 'Optimising accelerator for CAD workstation'	1-3, 11-18
Y	see page 219, paragraph 3.34 - page 220; figure 6 ---	4-7
Y	COMPUTER ARCHITECTURE NEWS vol. 17, no. 2, April 1989, NEW YORK US pages 15 - 25 G. SOHI AND S. VAJAPEYAM 'Tradeoffs in instruction format design for horizontal architectures' see page 15, left column, line 14 - line 20 --- -/--	6,7

° Special categories of cited documents : [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

3

IV. CERTIFICATION

Date of the Actual Completion of the International Search 23 JUNE 1993	Date of Mailing of this International Search Report 07. 07. 93
International Searching Authority EUROPEAN PATENT OFFICE	Signature of Authorized Officer MICHEL T.G.R.

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)

Category °	Citation of Document, with indication, where appropriate, of the relevant passages	Relevant to Claim No.
Y	FR,A,2 666 670 (J. C. LAWSON) 13 March 1992 see claim 1; figure 1	4,5
X	1988 IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS AND PROCESSORS 3 October 1988, NEW YORK, USA pages 582 - 585 J. KIM ET AL 'A novel VLSI architecture for the real-time implementation of 2-D signal processing systems'	1-3,5,8
A	see page 583, right column, line 5 - page 584, left column, line 28; figure 2	12-18
A	WO,A,9 119 259 (ADAPTIVE SOLUTIONS INC) 12 December 1991 see abstract; claim 1; figure 2	1-3,6,7
A	MICROPROCESSING AND MICROPROGRAMMING no. 26, 1989, pages 139 - 152 P. ROUNCE 'VLSI architecture research within the ESPRIT SPAN project' see page 148, left column, line 42 - right column, line 30	6,7,17
P,X	US,A,5 163 133 (N. MORGAN AND A. GEVINS) 10 November 1992 see the whole document	1-3,8, 12-14,18

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON INTERNATIONAL PATENT APPLICATION NO. GB 9300580 SA 71459

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information. 23/06/93

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
FR-A-2666670	13-03-92	None	
WO-A-9119259	12-12-91	EP-A- 0485466	20-05-92
US-A-5163133	10-11-92	None	

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82